

21世纪全国本科院校电气信息类创新型应用人才培养规划教材

微控制器原理及应用

主编 丁筱玲 王成义





21世纪全国本科院校电气信息类创新型应用人才培养规划教材

微控制器原理及应用

主　编　丁筱玲　王成义
副主编　李天华　施国英　赵立新

北京大学出版社
PEKING UNIVERSITY PRESS

内 容 简 介

本书以汇编语言和 C 语言为编程语言，系统地介绍了 80C51 系列微控制器的组织结构、工作原理、指令系统、程序设计、各功能单元、外部串行扩展技术、典型外围接口技术和编程仿真工具 Proteus 及 Keil μVision4。本书原理与应用紧密结合；突出微控制器的基本原理、体系结构；重点介绍微控制器各个功能单元的工作原理和应用、串行扩展技术和外围接口技术。为适应传统与现代交互的教学方式，编程仿真工具及 C 语言在附录中介绍。

本书可作为高等院校电子信息科学与技术、电子信息工程、通信工程、自动化、电气工程、机电工程、计算机应用等专业的“微控制器原理及应用”“单片机原理及应用”等课程的教学用书，也可作为工程技术人员、微控制器爱好者的技术参考书。

图书在版编目(CIP)数据

微控制器原理及应用/丁筱玲，王成义主编. —北京：北京大学出版社，2014.5

(21 世纪全国本科院校电气信息类创新型应用人才培养规划教材)

ISBN 978-7-301-24812-6

Ⅰ.①微…　Ⅱ.①丁…②王…　Ⅲ.①微控制器—高等学校—教材　Ⅳ.①TP332.3

中国版本图书馆 CIP 数据核字(2014)第 212650 号

书　　　名：微控制器原理及应用

著作责任者：丁筱玲　王成义　主编

策 划 编 辑：童君鑫

责 任 编 辑：宋亚玲

标 准 书 号：ISBN 978-7-301-24812-6/TH・0408

出 版 发 行：北京大学出版社

地　　　址：北京市海淀区成府路 205 号　100871

网　　　址：http://www.pup.cn　新浪官方微博：@北京大学出版社

电 子 信 箱：pup_6@163.com

电　　　话：邮购部 62752015　发行部 62750672　编辑部 62750667　出版部 62754962

印　刷　者：

经　销　者：新华书店

787 毫米×1092 毫米　16 开本　19.25 印张　450 千字

2014 年 5 月第 1 版　2014 年 5 月第 1 次印刷

定　　　价：42.00 元

前　　言

人类社会进入 21 世纪以来，科学技术飞速发展，信息技术更是一日千里。这其中，物联网、智能家居、无线传感网络、智慧城市等概念已经逐渐进入普通百姓的生活。在这些技术和应用中，微控制器(Microcontroller)占据着举足轻重的地位，可以这样说，没有微控制器，就没有这些技术和应用，微控制器是基石。

微控制器种类繁多，从数据总线上划分，有 8 位、16 位、32 位；从指令架构上划分，有 CISC 架构、RISC 架构；从存储器结构上划分，有哈佛结构、冯•诺依曼结构。考虑到初学者的因素，本书选择 80C51 系列微控制器来讲授。80C51 系列微控制器是 8 位、CISC 架构、哈佛结构的微控制器，它的资源非常丰富，性价比高，简单易学，在我国应用最为广泛。

本书作者有多年讲授 80C51 系列微控制器的经验，理论知识和实践经验丰富。本书具有如下特色。

1. 适合多种学时教学。由于学科、专业的不同，各个学校不同专业的教学学时不同。本书基本架构按照传统方式编写，先讲微控制器的基本结构，接着介绍指令系统及汇编编程，再介绍标准 80C51 的片内功能单元，最后介绍 80C51 的扩展和接口电路。授课教师可用较少的学时完成教学任务。附录 A Keil µVision4 集成开发环境与 C 语言程序设计及附录 B Proteus 仿真设计可让学生自己学习。如果学时较多，可根据实际需要将附录 A、附录 B 插入到课程体系中一并讲解。

2. 既适合教学，也适合读者自学。对于微控制器的爱好者来说，大部分人已经掌握了标准的 C 语言编程。此时，可用较短时间学习微控制器的基本结构、指令系统、80C51 的片内功能单元、80C51 的扩展和接口电路；而片内功能单元及扩展和接口的汇编编程方法可暂不学习。当学习并熟知基本内容后，再重点研习附录 A Keil µVision4 集成开发环境与 C 语言程序设计及附录 B Proteus 仿真设计，以达到迅速掌握微控制器开发编程工具、提高自己动手实践能力的目的。

3. 汇编语言与 C 语言结合。书中基本原理部分采用汇编语言讲解，培养学生使用汇编语言思考问题的能力，而片内功能单元部分既讲解汇编语言编程方法，又讲解 C 语言编程方法，让学生清楚汇编语言编程和 C 语言编程两种方式的共同点和不同点，以加深理解。在微控制器的扩展和外围接口方面，只提供汇编语言的编程方式，不提供 C 语言编程方式，培养学生自己使用 C 语言编程的能力，同时培养学生使用互联网查找资料，解决 C 语言编程中遇到的问题的能力。

4. Keil µVision4 与 Proteus 相结合。Keil µVision4 集成开发环境是 ARM 公司下属公司 Keil 公司推出的 80C51 系列微控制器开发工具，是当前开发 80C51 系列微控制器的主要工具。它功能强大，汇编语言开发和 C 语言开发都支持。Proteus 软件是英国 Labcenter Electronics 公司出版的 EDA 工具软件，它元器件库丰富且简单易用，不但具有其他 EDA

工具软件的仿真功能，而且能仿真 MCU 及外围器件，是目前最好的仿真 MCU 及外围器件的工具。以上两者的结合，可搭建一个虚拟的微控制器实验室，完成绝大部分的实验内容。

全书主体内容共分 6 章。第 1 章为绪论，介绍微控制器的基础知识；第 2 章为 80C51 系列微控制器的片内基本结构，介绍 80C51 系列微控制器的内部结构、引脚功能、存储器结构及工作方式等；第 3 章为 80C51 系列微控制器的指令系统及程序设计；第 4 章为 80C51 系列微控制器的功能单元，介绍端口结构、定时器/计数器、中断系统、串行通信功能；第 5 章为微控制器的外部串行扩展技术，介绍 I^2C 扩展技术和 SPI 扩展技术；第 6 章为微控制器的典型外围接口技术，介绍按键和显示技术，ADC 和 DAC 技术。书后附有 Keil μVision4 集成开发环境与 C 语言程序设计、Proteus 仿真设计、ASCII 码表、80C51 系列微控制器指令系统表，便于读者学习查阅。

本书第 1、2 章由丁筱玲负责编写，第 3、4 章由王成义负责编写，第 5、6 章由李天华负责编写，附录 A 由赵立新负责编写，附录 B、C、D 由施国英负责编写。全书由丁筱玲、王成义统编及统校，书中图表由李天华、施国英、赵立新分工制作，研究生吴玉红、杨翠翠、朱瞳、张昆等参与整理部分材料。

感谢深圳风标科技公司(Proteus 中国大陆总代理)的支持。

在本书编写、出版过程中，作者借鉴了许多优秀教材和技术专家的宝贵经验和技术资料，在此一并表示诚挚的谢意。

由于作者水平有限，书中不妥之处难以避免，敬请读者批评指正。

丁筱玲

2014 年 3 月

目　录

第 1 章 绪 论

本章教学要点

知识要点	掌握程度	相关知识
关于微控制器	掌握微控制器的概念； 了解微控制器的历史； 了解微控制器的发展趋势	单片微型计算机、MCU 分类； MCU 的四个发展阶段； 多功能、高速低耗、结构兼容
微控制器的特点及应用	熟悉微控制器的特点； 掌握微控制器的应用	体小、价低、方便、稳定可靠； 家电、智能仪表、武器装备
80C51 系列微控制器	掌握 80C51 系列微控制器的概念； 熟悉 80C51 系列芯片种类、特点； 熟悉近年流行的几个系列的 MCU	8 位 MCU 的典型代表； 以 80C51 为核心、与 MCS-51 兼容芯片系列：AT89C5X、STC89C、STC12C 及 C8051F 等



导入案例

“神九”“神八”归来，带给我们什么



2012年6月16日傍晚，“神九”载人飞船于酒泉卫星发射中心中国载人航天发射场成功发射升空，执行与天宫一号首次载人交会对接任务。中国首位女航天员刘洋和“神九”飞行乘组指令长景海鹏、手动控制交会对接“驾驶员”刘旺一道飞天，并于北京时间6月29日10时03分，在内蒙古中部草原主着陆场预定区域顺利返回。在13天的太空飞行中，3名航天员圆满开展进驻天宫一号、首次手控交会对接、航天医学和空间实验等一系列太空工作与生活。

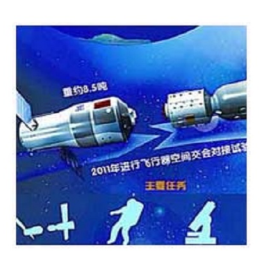
伴随“神九”“神八”的归来，关于交会对接、载人航天的话题远未尘埃落定，与之前的“神五”“神六”“神七”工程相比，“太空之吻”虽同样万众瞩目，但任务成功后类似“举国欢庆”的场面已经不再复现。归来的“神九”“神八”，面对的是人们更加习以为常的目光，甚至还有一些质疑。



种种疑虑，其来有自。老百姓的眼界日益开阔，不再只满足于围观天上的热闹，更要追问一声门道。遥远太空中的“太空之吻”无论加上怎样浪漫的想象，似乎也与当下纷扰杂沓的日常生活相距甚远。那么，载人航天真的无法与我们的现实生活有任何关系吗？

载人航天工程总设计师周建平说，中国载人航天20年的花费“不及美国一年的投入”，从横向比较上来看，中国载人航天不算“烧钱”。但即便如此，仍然有很多人在疑惑，这钱到底花得有什么意义呢？归根到底，不是“烧钱”“不烧钱”的问题，而是“花在哪儿”“值不值”？

直观上说，载人航天“看得见”的成果，就是正在运行的天宫一号空间实验室和实验室里搭载的各种科学实验。而透过这些直观的成果，一次载人航天实验，背后是成百上千项科技创新，是技术更新引领的产业升级，是各种相关产业的直接、间接受益，是一批又一批人才的崛起。



小至婴儿纸尿裤、纯净水净化技术，大至卫星导航系统、数控系统，都是缘起于航天科技，随着时间的推移，技术进步、成本下降慢慢“走下神坛”，成为人们日常生活中不可或缺的东西。现今尖端的航天技术，未来将对人们的生活产生哪些影响仍未可尽知，正如20世纪60年代应美国军方计划而诞生的互联网，在进入民用领域后对人类社会产生的深远影响，在当时又有几个人曾

料想到呢？除了这些眼前和长远的物质成果之外，每一次载人航天的成功，无形中又是一次全民科普，一次对追求科学精神的唤醒。

康德有云：“有两样东西，我们愈经常愈持久地加以思索，它们就愈使心灵充满日新又新、有加无已的景仰和敬畏：在我之上的星空和居我心中的道德法则”。哲人追索星空，是欲寻觅心灵栖居之所。而人类目前进行的太空探索，则有着更加现实的目标。地球资源的日渐匮乏，催促着人们探索新的家园。然而探索新家园的“门票”就如诺亚方舟一样，并不是谁都可以挤得上。曾经在过去百余年间错过太多的我们，此刻是要紧紧追上，还是要再次被落下？

1.1 什么是微控制器(单片机)

微控制器(单片机)就是在一块半导体硅片上集成了微处理器(CPU)、存储器(RAM、ROM)和各种功能单元(定时器/计数器、并行 I/O 口、串行口、ADC 等)的集成电路芯片。这样一块集成电路芯片具有一台计算机的属性，因而被称为单片微型计算机(Single Chip Microcomputer)，简称单片机。为了更好地体现其控制应用，在国际上，一般把单片机称为微控制器(Micro Controller Unit，MCU)。图 1-1 所示为较常用的 STC12C5A60S2 微控制器芯片。



图 1-1 微控制器芯片

MCU 按照其用途可以分为通用型和专用型两大类。

通用型 MCU 具有比较丰富的内部资源，性能全面且适应性强，可满足多种应用需求。通用型 MCU 把可开发的内部资源，如 RAM、ROM、I/O 等功能部件全部提供给用户。用户可以根据具体需求，充分利用 MCU 内部资源，设计一个以通用 MCU 芯片为核心，再配以外部接口电路及其他外围设备来满足不同需要的测量控制系统。

专用型 MCU 是专门针对某些产品的特定用途而制作的 MCU，如打印机、家用电器、健身器材以及各种通信设备中的专用 MCU。这种应用的最大特点是针对性强且数量巨大。因此，MCU 芯片制造商常与产品厂家合作，设计专用 MCU 芯片，以最大限度地降低成本。但是无论多么“专用”，MCU 的核心地位是不变的。

1.2 微控制器的发展历史

MCU 的发展历史大致可分为四个阶段。

第一阶段(1974—1976 年)：MCU 初级阶段。因工艺限制，MCU 采用双片的形式，而且功能比较简单。

第二阶段(1976—1978 年)：低性能 MCU 阶段。以 Intel 公司制造的 MCS-48 系列 MCU 为代表，它集成了 8 位 CPU、并行 I/O 口、8 位定时器/计数器、RAM、ROM，但没有串行接口，中断处理也比较简单，片内 RAM 和 ROM 容量都较小。

第三阶段(1978—1982 年)：高性能 MCU 阶段。典型代表有 Intel 公司制造的 MCS-51 系列 MCU、Freescale 公司(原 Motorola 公司的半导体事业部)的 6801 和 Zilog 公司的 Z8 等。这个阶段的 MCU 普遍带有串行接口、多级中断系统、16 位定时器/计数器，片内 RAM 和 ROM 容量加大，寻址范围可达 64KB。

第四阶段(1982 年至现在)：8 位 MCU 巩固发展，大量应用；16 位 MCU 逐渐占有一席之地；32 位 MCU 推出阶段。此阶段的主要特征是一方面开始发展 16 位 MCU、32 位 MCU 及专用型 MCU；另一方面不断完善高档 8 位 MCU，以满足不同的用户需求。对于 8 位 MCU 来说，生产厂家大量涌现，国内也出现了几家，并逐渐占领了低端市场。代表厂家为 ATMEL、NXP(原 Philips 的半导体事业部)、Silicon Lab、STC 等。对于 16 位 MCU 来说，代表厂家为 TI、Freescale、凌阳等。但随着 32 位 MCU 的推出，尤其是 ARM 公司 Cortex 系列 MCU 的推出，16 位 MCU 处于比较尴尬的境地。有专家认为，在 8 位和 32 位 MCU 的夹击下，16 位 MCU 将走向消亡。

尽管目前 8 位 MCU 种类繁多，但其中最具代表性的是 Intel 公司的 MCS-51 系列 MCU。80C51 系列 MCU 是在 MCS-51 的基础上于 20 世纪 80 年代发展起来的，其功能有很大增强。直到现在，80C51 系列 MCU 仍然是 8 位 MCU 的主流品种。本书将以 80C51 系列 MCU 为主，介绍微控制器的原理及其应用。

1.3 微控制器的发展趋势

从近 40 年 MCU 的发展历程可以看出，它正朝多功能、多选择、高速度、低功耗、低价格、扩大存储容量和加强 I/O 功能及结构兼容方向发展，图 1-2 所示为微控制器最小系统。微控制器今后的发展趋势为以下几个方面。

1. 多功能化

MCU 可集成越来越多的内置部件，常用的部件有：

(1) 存储器类，包括程序存储器 MROM/OTP ROM/EPROM/EEPROM/Flash ROM 和数据存储器 SRAM/SDRAM/SSRAM。

(2) 串行接口类，包括 UART、SPI、I2C、CAN、IR、Ethernet、HDLC。

(3) 并行接口类，包括 Centronics、PCI、IDE、GPIO 等。

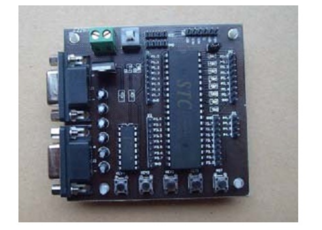


图 1-2　微控制器最小系统

(4) 定时和时钟类，包括定时器/计数器、实时时钟(RTC)、Watchdog、Clock out。

(5) 专用和外围接口类，包括 Comparer(比较器)、ADC、DAC、LCD 控制器、DMA、PWM、PLL、温度传感器等。

甚至有的 MCU，如 NS(National Semiconductor)公司的 MCU，已把语音、图像部件也集成到 MCU 中，目的就是在单个器件中集成所有需要用到的部件，构成片上系统(SoC，System on Chip)。Silicon Labs 公司推出的 C8051F 系列的 MCU 在一个芯片中集成了构成数据采集系统或控制系统所需要的几乎所有的数字和模拟外围接口和功能部件，这种混合信号芯片实质上已构成了混合信号片上系统(Mixed-Signal SoC)。

2. 多核化

随着嵌入式应用的深入，特别是在数字通信和网络中的应用，对处理器提出了更高的要求。为满足这种要求，现在已出现多核结构的处理器。

Freescale 公司研发的 MPC8260 PowerQUiCCⅡ就是一种先进的为电信和网络市场而设计的集成通信微处理器。它融合了两个 CPU——嵌入式 PowerPC 内核和通信处理模块(CPM)。由于 CPM 分担了嵌入式 PowerPC 核的外围工作任务，这种双处理器体系结构功耗反而要低于传统体系结构的处理器。

Infineon 公司推出的 TCl0GP 和增强型 TC1130 都是三核(TriCore)结构的微处理器。它同时具备 RISC、CISC 和 DSP 功能，是一种建立在 SoC 概念上的结构。这种 MCU 由三个核组成：MCU 和 DSP 核、数据和程序存储器核、外围专用集成电路(ASIC)。这种 MCU 的最大特点是把 DSP 和 MCU 融合成一个单内核，大大提高了 MCU 的功能。具有类似结构的还有 Hitachi 公司的 SH7410、SH7612 等；它们用于既需要 MCU 又需要 DSP 功能的场合，比单独使用 MCU 和 DSP 的组合拥有更优越的性能。

3. 低功耗化

现在新推出的 MCU 的功耗越来越低，很多 MCU 都有多种工作方式，包括等待、暂停、休眠、空闲、节电等工作方式。例如 NXP 的 P87LPC762，空闲状态下的电流为 1.5mA，而在节电方式下电流只有 0.5mA。很多 MCU 还允许在低振荡频率下以极低的功耗工作。例如，P87LPC764 在 32768Hz 低频下，正常工作电流仅为 I_{dd}=16μA(V_{dd}=3.6V)，空闲模式下 I_{dd}=7μA(V_{dd}=3.6V)。

4. 宽工作电压

扩大电源电压范围以及在较低电压下仍能工作是现在新推出的 MCU 的一个特点。目前一般 MCU 都可以在 3.3～5.5V 的范围内工作，有些产品则可以在 2.2～6V 的范围内工作。例如，Fujitsu 公司的 MB8919X、MB8912X 和 MB89130 系列及 F2MC-8L 系列的 MCU，绝大多数工作电压范围都为 2.2～6V；而 TI 公司的 MSP430X11X 系列的 MCU 的工作电压可以低达 2.2V。Freescale 公司针对长时间处于待机模式的装置所设计的超省电 HCS08 系列 MCU，已经把可工作的最低电压降到了 1.8V。

5. 封装小型化

现在 MCU 的封装水平已大大提高，有越来越多的 MCU 采用了各种贴片封装形式，以满足便携式手持设备的需要。Microchip 公司推出了目前世界上体积最小的 6 引脚 PIC10F2XX 系列 MCU。为了适应各种应用的需要，减少驱动电路，很多 MCU 的输出能力都有了很大提高，Freesale 公司的 MCU 的 I/O 口灌电流可达 8mA 以上，而 Microchip 公司的 MCU 可达 20～25mA，其他如 AMD、Fujitsu、NEC、Infineon、Hitachi、Atmel、Toshiba 等公司的都在 8～20mA 之间。

6. 低噪声布线技术

在过去一般 MCU 中，电源与地引脚是安排在芯片封装的对角上，即左上、右下或右上、左下位置上。这种安排会使电源噪声对 MCU 的内部电路造成的干扰相对较大。现在很多 MCU 都把电源和地引脚安排在两个相邻的引脚上。这样既降低了干扰，又便于在印制电路板上对去耦电容器进行布线，降低系统的噪声，如 STC15F2K60S2。

1.4 微控制器的特点及应用

1.4.1 微控制器的特点

MCU 的出现是微型计算机技术高速发展的产物。MCU 体积小、价格低、应用方便、稳定可靠，所以，MCU 的发展和普及给工业自动化等领域带来了一场重大革命。仅从体积小方面来说，MCU 几乎可以在任何设备或任意装置上做成非常小的、功能比较完善的 MCU 嵌入式系统而置于其中，以实现各种方式的检测、计算或控制。在这一点上，一般的微型计算机根本做不到。由于 MCU 本身就是一个计算机系统，因此只要在 MCU 的外部适当增加一些必要的外围扩展接口电路，就可以灵活地构成各种应用系统，如工业自动检测监视系统、数据采集系统、自动控制系统、智能仪器仪表等。

MCU 之所以应用如此广泛，主要原因是以 MCU 为核心构成的应用系统具有以下优点。

(1) 功能齐全，应用可靠，抗干扰能力强。

(2) 简单方便，易于普及。由于 MCU 技术是一门较为容易掌握的技术，故 MCU 应用系统设计、组装、调试已经是十分简单，广大工程技术人员通过学习可以很快地掌握其应用技术。

(3) 发展迅速，前景广阔。在短短几十年的时间里，MCU 就经过了 4 位机、8 位机、16 位机、32 位机等几大发展阶段。尤其是形式多样、集成度高、功能日臻完善的 MCU 不



断问世，更使得 MCU 在工业控制及工业自动化领域获得长足的发展和广泛应用。近几年，MCU 的内部结构愈加完美，配套的片内外围功能器件越来越完善，为应用系统向更高层次和更大规模的发展奠定了坚实的基础。

(4) 嵌入容易，用途广泛。MCU 的体积小、性价比高、应用灵活性强。在 MCU 出现以前，人们要想制作一套自动控制系统，往往采用大量的模拟电路、数字电路、分立元件来完成，以实现计算、判断和控制功能。这样，不仅系统的体积庞大，而且因为线路复杂及连接点太多，极易出现故障。MCU 出现以后，电路的组成和控制方式都发生很大的变化。在 MCU 应用系统中，这些控制功能的部分都已经由 MCU 的软件程序实现，其他电子线路则由片内的外围接口电路来替代。原来必须由电子线路实现的计算、比较、判断等功能现在已经由 MCU 和软件及片内的外围接口电路来取代。

1.4.2 微控制器的应用

MCU 主要应用在检测、控制领域，它具有小巧灵活、成本低、可靠性好、适应温度范围宽、易扩展等特点。以下是 MCU 应用领域的举例。

1. 家用电器领域

目前国内家用电器已普遍采用 MCU 控制取代传统的控制电路，如洗衣机、电冰箱、空调、微波炉、电饭煲、电视机、录像机及其他视频音像设备的控制器。

2. 办公自动化领域

现代办公室中所使用的大量通信、信息产品多数都采用了 MCU，如通用计算机系统中的键盘译码、磁盘驱动、打印机、绘图仪、复印机、电话、传真机、考勤机等。

3. 商业营销领域

在商业营销领域已广泛使用的电子秤、收款机、条形码阅读器、仓储安全监测系统、商场保安系统、空气调节系统、冷冻保鲜系统中，已纷纷采用 MCU 构成专用系统，这主要是由于这种系统有明显的抗病菌侵害、抗电磁干扰等高可靠性能的保证。

4. 工业自动化

工业过程控制、过程监测、工业控制器及机电一体化控制系统等这些系统除一些小型工控机之外，许多都是由 MCU 为核心的单机或多机网络系统。例如，工业机器人的控制系统是由中央控制器、感觉系统、行走系统、擒拿系统等节点构成的多机网络系统。

5. 智能仪表与传感器网络

目前各种变送器、电气测量仪表普遍采用 MCU 应用系统替代传统的测量系统，使测量系统具有各种智能化功能，如存储、数据处理、查找、判断、联网和语音功能等。

将 MCU 与传感器相结合可以构成新一代的智能传感器，它将传感器初级变换后的电量作进一步的变换、处理，输出能满足远距离传送、能与微机接口的数字信号。

6. 汽车电子设备

MCU 已经广泛地应用于各种汽车电子设备中，如汽车安全系统、智能自动驾驶系统、汽车集中显示系统、卫星汽车导航系统、汽车防撞监控系统、汽车自动诊断系统及汽车黑匣子等。

7. 武器装备

在现代化的武器装备中，如飞机、军舰、大炮、坦克、导弹、鱼类制导、智能武器装备、航天飞机导航系统，都有 MCU 作为主控芯片嵌入其中，发挥着重要作用。

综上所述，MCU 应用的意义绝不限于它的功能及所带来的经济效益，更重要的意义在于，MCU 的应用正从根本上改变着传统的控制系统设计思想和设计方法。从前必须由模拟电路或数字电路实现的大部分控制功能，现在已能使用 MCU 通过软件方法实现了，这种以软件取代硬件并能提高系统性能的控制技术，称之为微控制技术。这标志着一种全新概念的建立。随着 MCU 应用技术的推广普及，微控制技术必将不断发展，日益完善。

1.5　80C51 系列微控制器简介

80C51 系列 MCU 是在 MCS-51 系列的基础上发展起来的。早期的 80C31 只是 MCS-51 系列众多芯片中的一类，但是随着后来的发展，80C51 MCU 已经形成独立的系列，并且成为当前 8 位 MCU 的典型代表。

MCS-51 的原生产厂是 Intel 公司，最早推出 80C51 芯片的也是 Intel 公司，并且作为 MCS-51 的一部分，按原 MCS-51 芯片的规则命名，如 80C31、80C51、87C51 和 89C51，这样我们就能很容易地认识 80C51 的系列芯片。但是后来有越来越多的厂商生产 80C51 系列芯片，如 NXP、Atmel、Silicon Labs、SST、STC 及华邦等公司。这些芯片都是以 80C51 为核心并且与 MCS-5l 芯片兼容，但它们又各具特点。然而由于生产厂家众多，芯片的类型也很多，为了体现各公司的特色，芯片的命名便不再遵循统一的命名规则，这也给我们对 MCU 型号的认知带来了麻烦。例如，STC 公司生产的 STC12C 系列的 STC12C5410AD MCU，我们无法使用 Intel 的命名规则来判断其所含有的资源(如 ROM、RAM 的数量)，当然，此命名也有规律可循；再比如 Silicon Labs 公司生产的 MCU C8051F410，其命名规律更难以琢磨，我们几乎无法从型号上发现其所含有的资源。

新一代 80C51 的兼容芯片，还在芯片中增加了一些外部接口功能单元，如 ADC、DAC、可编程计数器阵列(PCA)、监视定时器(WDT)、高速 I/O 口、计数器的捕获/比较逻辑等。有些公司(如 Silicon Labs)还增加了片上调试功能，在调试时所有的数字和模拟外设都能正常工作，实时反映了 MCU 真实的运行情况。所有这些新一代的兼容芯片已开始在我国使用，其中尤以我国的 STC 公司的芯片及国外的 Atmel、Silicon Labs、NXP 等公司同名芯片及其衍生产品最受欢迎。

下面介绍最近十几年在我国比较流行的几个系列的 MCU。

1. Atmel 公司生产的 AT89C5X 系列 MCU

Atmel 是成立于 1984 年的一家美国半导体公司，是世界上高级半导体产品设计、制造和行销的领先者，产品包括了微处理器、可编程逻辑器件、非易失性存储器、安全芯片、混合信号及 RF 射频集成电路。该公司于 1994 年以 E^2PROM 技术与 Intel 公司的 80C51 内核的使用权进行交换，获得了制造基于 80C51 内核 MCU 的技术。Atmel 公司将 Flash 技术

与 80C51 内核相结合，形成了独具特色的以 Flash ROM 为程序存储器的 AT89C5X 系列 MCU。

AT89C5X 系列 MCU 继承了 80C51 的全部功能，在引脚以及指令系统方面完全兼容。此外，AT89C5X 系列 MCU 中的某些型号又增加了一些新的功能，如 WDT、ISP 及 SPI 串行总线技术等。其中 AT89S51 增加了 ISP 及 SPI，其时钟频率最高达 33MHz，Flash 存储器不但允许并行重复编程，还支持在线可编程写入技术，串行写入、速度更快、稳定性更好，烧写电压也仅需要 4～5V。另外，AT89S51 也支持由软件选择的两种低功耗方式，非常适合于电池供电或其他要求低功耗的场合。

表 1-1 为 AT89C5X 系列 MCU 的主要产品片内硬件资源。

表 1-1 Atmel 公司生产的 AT89C5X 系列 MCU 主要产品的片内硬件资源

型号	Flash ROM (KB)	RAM (B)	I/O(位)	T/C (个)	中断源 (个)	管脚数目 (个)
89C1051	1	128	15	1	3	20
89C2051	2	128	15	2	5	20
89C51	4	128	32	2	6	40
89C52	8	256	32	3	8	40
89C55	20	256	32	3	8	40
89C51RD	64	2048	32	4	9	44
89S2051	2	256	15	2	6	20
89S4051	4	256	15	2	6	20
89S51	4	128	32	2	6	40
89S52	8	256	32	3	8	40

2. STC 公司生产的 STC89C 系列及 STC12C 系列 MCU

STC(深圳宏晶科技有限公司)是深圳的一家 8051 系列 MCU 设计生产公司，STC 系列的 MCU 在中国的 80C51 系列 MCU 市场上占有较大比例。STC 现已成长为全球最大的 8051 系列 MCU 设计公司，现提供专用 MCU 设计服务，并致力于提供处于业内领先地位的、高性能 STC 系列 MCU 和 SRAM。其产品已通过国际权威认证机构 SGS(瑞士通用公证行)的多项认证(①EFT 测试认证：过 4kV 快速脉冲干扰；②绿色环保认证：无铅认证)。

(1) STC89C 系列 MCU 性能特点：最高工作频率为 80MHz，Flash 程序储存器容量为 4～64KB，RAM 数据储存器容量 512～1280B，内部集成 E^2PROM 2～16KB 及看门狗和专用复位电路，部分集成 ADC。

表 1-2 为 STC89C 系列 MCU 的主要产品片内硬件资源。

表 1-2　STC 公司生产的 STC89C 系列 MCU 主要产品的片内硬件资源

型　　号	Flash ROM (KB)	RAM (B)	E^2PROM(KB)	I/O (位)	T/C (个)	中断源 (个)	管脚数目 (个)
STC89C51RC	4	512	2	32/36	3	8	40
STC89C52RC	8	512	2	32/36	3	8	40
STC89C53RC	15	512	—	32/36	3	8	40
STC89C54RD+	16	1280	16	32/36	3	8	40
STC89C55RD+	20	1280	16	32/36	3	8	40
STC89C58RD+	32	1280	16	32/36	3	8	40
STC89C516RD+	64	1280	—	32/36	3	8	40

(2) STC12C 系列 MCU 性能特点：单时钟/机器周期，超小封装，2～4 路 PWM，8～10 位高速 ADC，Flash 程序储存器 512B～12KB，RAM 数据储存器 256～512B，集成 1KB 的 E^2PROM 及硬件 WDT。产品都有低功耗功能，都有 ISP 和 IAP 功能，以及强抗干扰和降低 EMI 性能。

表 1-3 为 STC12C 系列 MCU 的主要产品片内硬件资源。

表 1-3　STC 公司生产的 STC12C 系列 MCU 主要产品的片内硬件资源

型号	Flash ROM (KB)	RAM (B)	E^2PROM(KB)	I/O (位)	SPI (个)	A/D (位)	PCA/PWM (个)	管脚数目 (个)
STC12C2052	2	256	1	15	1	—	4	20
STC12C2052AD	2	256	1	15	1	8	4	20
STC12C4052	4	256	1	15	1	—	4	20
STC12C4052AD	4	256	1	15	1	8	4	20
STC12C5404	4	512	2	27/23	1	—	8	20/28/32
STC12C5404AD	4	512	2	27/23	1	10	8	20/28/32
STC12C5408	8	512	2	27/23	1	—	8	20/28/32
STC12C5408AD	8	512	2	27/23	1	10	8	20/28/32
STC12C5412	12	512	2	27/23	1	—	8	20/28/32
STC12C5412AD	12	512	2	27/23	1	10	8	20/28/32

STC 还生产有其他系列 MCU，如 STC90C 系列、STC11F 系列、STC15F 系列等。

3. Silicon Labs 公司生产的 C8051F 系列 MCU

C8051F 系列 MCU 是美国德克萨斯州的 Cygnal 公司设计和制造的混合信号片上系统 MCU，该公司于 2003 年并入 Silicon Labs 公司，后者更新原有 80C51 系列 MCU 结构，设计了具有自主知识产权的 CIP-51 内核的新 C8051F 系列 MCU。C8051F 系列 MCU 的主要模块包括模拟外设、片内 JTAG 调试和边界扫描、高速控制器内核、数字外设等几个部分。C8051F 系列 MCU 是目前 8 位 MCU 中功能最齐全、性能最优的一种，在科研生产上获得了广泛的应用。

C8051F 有多个系列，如 C8051F02X 系列、C8051F04X 系列、C8051F06X 系列、C8051F34X 系列、C8051F35X 系列、C8051F41X 系列等。我们以 C8051F04X 系列 MCU 为例进行介绍。C8051F04X 系列是完全集成的混合信号片上系统型 MCU，具有 64 个数字 I/O 引脚(C8051F040/2/4/6)或 32 个数字 I/O 引脚(C8051F041/3/5/7)，片内集成了一个 CAN2.0B 控制器。下面给出了 C8051F04X 系列 MCU 的一些主要特性：

(1) 高速、流水线结构的 80C51 兼容的 CIP-51 内核(可达 25MIPS)。

(2) 集成 CAN2.0B 控制器，具有 32 个消息对象，每个消息对象有其自己的标识。

(3) 全速、非侵入式的系统调试接口(片内)。

(4) 真正 12 位(C8051F040/1)或 10 位(C8051F042/3/4/5/6/7)、100ksps 的 ADC，带 PGA 和 8 通道模拟多路开关。

(5) 允许高电压差分放大器输入到 12/10 位 ADC(60V 峰-峰值)，增益可编程。

(6) 真正 8 位 500ksps 的 ADC，带 PGA 和 8 通道模拟多路开关(C8051F040/1/2/3)。

(7) 两个 12 位 DAC，具有可编程数据更新方式(C8051F040/1/2/3)。

(8) 64KB(C8051F040/1/2/3/4/5)或 32KB(C8051F046/7)可在系统编程的 Flash ROM。

(9) 4352(4KB+256B)B 的片内 RAM。

(10) 可寻址 64KB 地址空间的外部数据存储器接口。

(11) 硬件实现的 SPI、SMBus/I^2C 和两个 UART 串行接口。

(12) 5 个通用的 16 位定时器。

(13) 具有 6 个捕捉/比较模块的可编程计数器/定时器阵列。

(14) 片内看门狗定时器、VDD 监视器和温度传感器。

具有片内 VDD 监视器、看门狗定时器和时钟振荡器的 C8051F04X 系列器件是真正能独立工作的片上系统。所有模拟和数字外设均可由用户固件使能/禁止和配置。Flash ROM 还具有在系统重新编程能力，可用于非易失性数据存储，并允许现场更新固件。

片内 JTAG 调试电路允许使用安装在最终应用系统产品上的 MCU 进行非侵入式(不占用片内资源)、全速、在系统调试。该调试系统支持观察和修改存储器和寄存器，支持断点、观察点、单步及运行和停机命令。在使用 JTAG 调试时，所有的模拟和数字外设都可全功能运行。

每个 MCU 都可在工业温度范围(−45～+85℃)工作，工作电压为 2.7～3.6V。端口 I/O、$\overline{\text{RST}}$ 和 JTAG 引脚都容许 5V 的输入信号电压。C8051F040/2/4/6 为 100 管脚 TQFP 封装，C8051F041/3/5/7 为 64 管脚 TQFP 封装。

表 1-4 为 C8051F04X 系列 MCU 的主要产品片内硬件资源。

表 1-4　Silicon Labs 公司生产的 C8051F04X 系列 MCU 的片内硬件资源

	MIPS(峰值)	Flash 存储器(KB)	RAM	外部存储器接口	SMBus/I^2C 和 SPI	CAN	UART	定时器(16 位)	可编程计数器阵列	数字端口 I/O	12 位 100ksps ADC 输入	10 位 100ksps ADC 输入	8 位 100ksps ADC	高电压差分放大器	电压基准	温度传感器	DAC 分辨率(位)	DAC 输出	电压比较器	封装
C8051 F040	25	64	4352	√	√	√	2	5	√	64	√	—	8	√	√	√	12	2	3	100TQFP
C8051 F041	25	64	4352	√	√	√	2	5	√	32	√	—	8	√	√	√	12	2	3	64TQFP
C8051 F042	25	64	4352	√	√	√	2	5	√	64	—	√	8	√	√	√	12	2	3	100TQFP
C8051 F043	25	64	4352	√	√	√	2	5	√	32	—	√	8	√	√	√	12	2	3	64TQFP
C8051 F044	25	64	4352	√	√	√	2	5	√	64	—	√		√	√	√	—	—	3	100TQFP
C8051 F045	25	64	4352	√	√	√	2	5	√	32	—	√		√	√	√	—	—	3	64TQFP
C8051 F046	25	32	4352	√	√	√	2	5	√	64	—	√		√	√	√	—	—	3	100TQFP
C8051 F047	25	32	4352	√	√	√	2	5	√	32	—	√		√	√	√	—	—	3	64TQFP

本 章 小 结

利用微控制器进行控制的技术称为微控制技术。微控制技术从根本上改变了传统的控制系统设计思想，它通过对单片机编程的方式代替由模拟电路或数字电路实现的大部分控制功能，是对传统控制的一次革命。

传统控制系统的控制功能是通过电器元件和线路连接等硬件手段实现的，一经完成，功能很难更改。若要改变功能，必须重新连接电路，十分不便。而微控制技术的控制是由硬件和软件共同实现的。只要改变程序的内容就可以在硬件线路基本功能的基础上实现多种功能。例如彩灯的控制，若由传统控制系统实现，则线路完成之后，彩灯的闪烁变换方式也就确定了；而若由单片机系统控制，不改变线路连接，只简单改变程序即可实现多种不同的彩灯闪烁方式。

80C51 系列 MCU 品种繁多，广泛应用于各行各业。因此，在应用中需要设计者对各种 MCU 都要了解，以便确定最佳的性价比，也就是说要能应用各种 MCU 进行设计。然而同时学习各种 MCU 的软硬件知识不仅难度较大而且没有必要。通常的方法是学习一种典型的 MCU 系列，掌握好其硬件结构和软件知识，在应用中，如果需要用到其他系列 MCU 时，只需将这两种系列的不同特点及软硬件上的不同之处稍加分析即可应用。

与其他类型的许多 MCU 相比较，80C51 系列 MCU 硬件结构简洁明了、特殊功能寄存器功能规范、软件指令系统易于掌握，是一种既便于讲授又便于学习、理解和掌握的 MCU。此外，这种 MCU 在国内介绍较多，资料比较齐全，其本身性能价格比较高，所以本书以 80C51 系列 MCU 为例，来介绍 MCU 知识。为了说明具有一定的针对性，我们以 Atmel 公司的 AT89C51(AT89S51)、AT89C52(AT89S52)和 STC 公司 STC89C51RC、STC89C52RC 为例，来介绍 MCU 的知识，也就是说，书中的程序可在这些 MCU 上调试通过，以增加学生的实践机会，提高学习兴趣。为了便于介绍，我们在书中统一使用 89C51 代表最基本的 4KB ROM 的 80C51 系列 MCU(AT89C51、AT89S51、STC89C51RC)，使用 89C52 代表扩展了 4KB ROM(共 8KB ROM)和定时器/计数器 T2(共有 3 个定时器/计数器)的 80C51 系列 MCU(AT89C52、AT89S52、STC89C52RC)。掌握了 80C51 系列 MCU 后，如果开发增强型的 51 系列或别的系列 MCU 应用系统，读者只需用很短时间，掌握相应 MCU 的特性和特点，即可用它来开发产品了。

本书是以大学本科生为教学对象编写，编写时考虑到同学们已经学过标准 C 语言、电路原理、模拟电子技术、数字电子技术和微机原理等基础课程。总的参考教学时数为 54 学时，其中包括课堂讲授和实验教学两部分。

思考题与习题

1．什么是微控制器？

2．80C51 系列 MCU 的主要特点是什么？

3．微控制器(MCU)和微处理器(MPU)有何不同？为什么说 MCU 是典型的嵌入式系统？

4．你认为在哪个领域中 MCU 的应用将得到较大发展？简述该领域的现状和技术发展趋势。

第 2 章

80C51 系列微控制器的片内基本结构

本章教学要点

知识要点	掌握程度	相关知识
微控制器引脚及接口	熟悉 80C51 微控制器各种引脚； 熟悉并行 I/O 口	电源及时钟引脚，控制引脚； P0 口、P1 口、P2 口、P3 口
80C51 微控制器 CPU 的结构	了解 80C51 微控制器 CPU 的结构	运算器； 控制器
80C51 微控制器的存储器结构	掌握 80C51 微控制器存储器结构	程序存储器； 数据存储器
时钟电路与 CPU 的工作时序	掌握时钟电路与 CPU 的工作时序	时钟电路； 时序定时单位
80C51 微控制器的工作方式	掌握 80C51 微控制器的工作方式	复位方式；程序执行方式； 低功耗方式；编程方式





导入案例

中国航母发展的战略地位与经济价值



发展航母的战略意义：关于航母，中国向来存在争议，有人以“中国是陆权国家”和“航母过时”等理由来反对中国建航母，还有人因担心“中国威胁”论会影响和周边国家的关系而反对。

中国发展航母会影响周边关系吗？某些国家散布“中国威胁论”的目的就是要阻止中国的军事现代化进程，在国际上制造对中国不利的舆论，遏制中国的发展。如果中国实力真的能达到让对手畏惧的程度，那么“中国威胁”自然消失。

世界霸权和恐怖主义的存在、发展对我们海外资源的安全构成严重威胁，主要由以下两个因素决定：一，霸权主义的性质就是通过控制世界的资源和对科技的垄断来达到控制世界的目的，我国要实现“和平崛起”的伟大战略目标必须要打破霸权规则，这不仅需要策略，更要军事实力做后盾。二，恐怖主义的蔓延。自美国“反恐”以来，恐怖活动日益猖獗。而我们在海外的能源获得主要就是海湾和非洲，海湾是恐怖的发源地，而非洲由于经济落后经常发生暴力事件，严重影响我国人员和能源安全。这就需要一只强大的远洋作战力量，一来可以对针对我国的恐怖活动进行打击，二来可以帮助友好国家打击分裂、暴力，保持社会稳定。



发展航母的必要性：现在美国最大的航母排水达到十万吨以上，各种飞机近百架，集各种先进技术于一身，拥有强大的电子、信息、反潜等手段。远、中、近三层防预，空中、水面、水下立体攻防，打击纵深可达数千公里，攻防兼备，成为美国推行霸权主义的中坚力量。中国由于长期受到“陆权”思维的误导，使海军发展缓慢，严重滞后于世界其他大国。而我们面对一些小国的挑衅投鼠忌器，总是采取妥协的方式，使我们的战略利益遭受严重损失。只要中国有两个重型航母群巡视南疆，马上就能令敌人束手，不战而屈人之兵。

发展航母的经济价值：以前有人担心发展太空计划会影响我国的经济发展，但事实却是带动了中国高科技的进步，促进了中国经济的增长。航母也是一样，它是一个系统工程，涉及装备制造、电子信息、新材料等多个领域，建航母必将带动我国民族工业的繁荣，促进产业升级，加速中华民族的复兴。



航空航天系统和国防军事、尖端武器等领域，微控制器的应用不言而喻。当然，微控制器应用的意义不仅在于它的广阔范围及所带来的经济效益。更重要的在于，它从根本上改变了控制系统传统的设计思想和设计方法。以前采用硬件电路实现的大部分控制功能，正在用微控制器通过软件方法来实现，以前自动控制中的PID调节，现在可以用微控制器实现具有智能化的数字计算控制、模糊控制和自适应控制。这种以软件取代硬件并能提高系统性能的控制技术称为“微控制”技术。

2.1　80C51系列微控制器的硬件组成

MCU的结构特征是将组成计算机的基本部件集成在一块电路芯片上，有的MCU生产厂家还集成了一些应用于测量和控制的特殊部件，从而构成了一个强大的计算机系统芯片。典型80C51系列MCU的基本组成结构如图2-1所示。

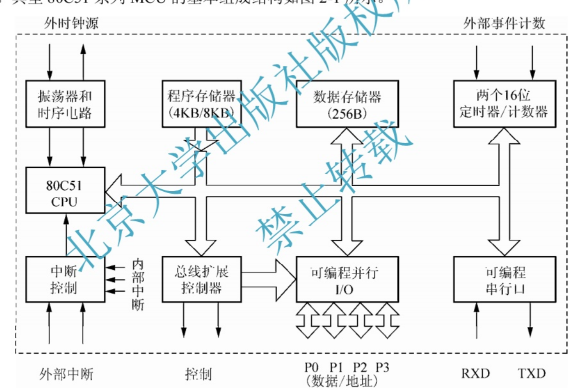


图2-1　典型80C51系列MCU的基本组成结构

下面分别对典型80C51系列MCU各组成部件进行简单介绍。

1. 微处理器(CPU)

80C51系列MCU中的中央处理器(CPU)是一个8位的处理器，和通用微处理器基本相同，同样包括了运算器和控制器两大部分，为了增强实时性，添加了“面向控制”的处理功能，如位处理、查表、多种跳转、乘除法运算、状态检测、中断处理等。



2. 程序存储器(ROM)

MCU 的存储器有两种基本结构：普林斯顿(Princeton)结构和哈佛(Harvard)结构。普林斯顿结构，又称为冯·诺依曼结构，是一种将程序指令存储器和数据存储器合并在一个存储空间的存储器结构，取指令和取操作数都在同一总线上，通过分时复用的方式进行；缺点是在高速运行时，不能达到同时取指令和取操作数，从而形成了传输过程的瓶颈。哈佛结构是一种将程序指令存储器和数据存储器分开的存储器结构，它的主要特点是将程序和数据存储在不同的存储空间中，即程序存储器和数据存储器是两个独立的存储器，每个存储器独立编址、独立访问，优点是减轻了程序运行时的访存瓶颈。考虑到 MCU“面向控制”的实际应用的特点和实时性的要求，80C51 系列 MCU 采用的结构是哈佛结构。

程序存储器，顾名思义，是用来存储程序的。89C51 内部集成 4KB 程序存储器，89C52 内部集成 8KB 程序存储器。由于技术的进步，现在的 80C51 系列 MCU 程序存储器均为 Flash ROM。如果片内程序存储器容量不够，在片外最多可将程序存储器扩充至 64KB；但实际上，由于技术的进步，80C51 系列 MCU 内部已可集成 64KB 的 Flash ROM。

3. 数据存储器(RAM)

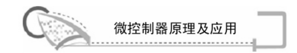
在 MCU 中，用随机存取存储器(RAM)来存储程序在运行期间的工作变量和数据，所以称为数据存储器。一般在 80C51 系列 MCU 内部设置一定容量(128～256B)的 RAM。这样，小容量的数据存储器以高速 RAM 的形式集成在 MCU 内，可以加快 MCU 运行的速度；而且这种结构的 RAM 还可以使存储器的功耗下降很多。在 80C51 系列 MCU 中，常把寄存器(如工作寄存器、特殊功能寄存器、堆栈等)在逻辑上划分在片内 RAM 空间中，所以可将 80C51 系列 MCU 内部 RAM 看成是寄存器堆，这样的结构也有利于运行速度的提高。对某些应用系统还可在外部扩展数据存储器。

4. 可编程并行 I/O 口

为了满足“面向控制”的实际应用的需要，MCU 提供了数量多、功能强、使用灵活的并行 I/O 口。不同 MCU 的并行 I/O 电路在结构上稍有差异。有些 MCU 的并行 I/O 口，不仅可灵活地选作输入/输出，而且还具有多种功能。例如，80C51 系列 MCU 的 P0 口，它既是 I/O 口，又是系统总线，从而为扩展外部存储器和 I/O 接口提供了方便，大大拓宽了 MCU 的应用范围。

5. 全双工串行口(UART)

现在的 80C51 系列 MCU 均配置了全双工串行口，有的 80C51 系列 MCU(Silicon Labs 公司的 MCU)甚至配置了多个串行口。串行口提供了与某些终端设备进行串行通信或和一些特殊功能的器件相连的能力，甚至可用多个 MCU 相连构成多机系统，使 MCU 的功能更强且应用更广。



6. 定时器/计数器(T/C)

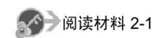
在 MCU 的实际应用中，往往需要精确的定时，或者对外部事件进行计数。为了减少软件开销和提高 MCU 的实时控制能力，现在的 MCU 均在内部配置了定时器/计数器电路，通过中断，实现定时/计数的自动处理。

7. 中断控制系统

中断控制系统的加入有效地解决了快速 CPU 与慢速外部设备之间的矛盾，可使 CPU 与外部设备并行工作，大大提高了工作效率；可以及时处理控制系统中许多随机产生的参数与信息，即计算机具有实时处理的能力，从而提高了控制系统的性能；使系统具备了处理故障的能力，提高了系统自身的可靠性。

8. 定时电路

计算机的整个工作是在时钟信号的驱动下，按照严格的时序有规律地执行各种操作。各种计算机均有自己的固定时序和定时电路。同样，80C51 系列 MCU 内部也设有定时电路，只需外接振荡元件即可工作。外接振荡元件一般选用晶体振荡器，或用廉价的 RC 振荡器作为振荡元件，也可使用外部时钟源。近来有的 80C51 系列 MCU(例如 STC 公司、Silicon Labs 公司的 MCU)将振荡元件也集成在芯片内部，这样不仅大大缩小了 MCU 的体积，同时也方便了使用。



串口 WiFi 模块 TLG10UA03

WiFi-TLG10UA03 是全新的一代嵌入式 UART-WiFi 模块产品，是基于 UART 接口的符合 WiFi 无线网络标准的嵌入式模块，内置无线网络协议 IEEE802.11 协议栈及 TCP/IP 协议栈，能够实现用户串口数据到无线网络之间的转换。通过 UART-WiFi 模块，传统的串口设备也能轻松接入无线网络。

WiFi-TLG10UA03 在前一代产品的基础上进行了全面的软硬件升级，功能更加强大，使用更加简单。全面支持串口透明数据传输模式，真正实现串口的即插即用。通信距离：室外 150m(11Mbit/s)、300m(1Mbit/s)；室内 30m(11Mbit/s)、100m(1Mbit/s)。

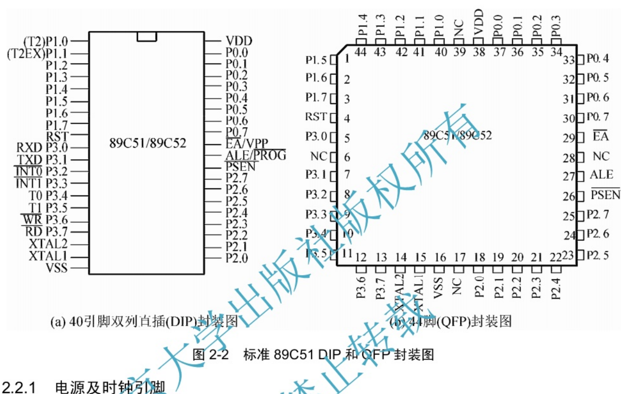
2.2 89C51 系列微控制器的引脚介绍

学习 MCU 首先必须知道怎样设计电路及怎样连线，这就需要了解 MCU 的引脚。熟悉并牢记各引脚的功能，是学好用好 MCU 的基本功。89C51 芯片是 DIP(Dual In-line Package)封装的，有 40 个引脚，如图 2-2(a)所示。89C52 芯片与 89C51 芯片封装形式一样。由于技术的进步及可靠性、小型化的要求，在应用中，现在的 MCU 出现了多种封装形式。例如，



STC 公司的 MCU，封装形式有 SOP(Small Out-line Package)、PQFP(Plastic Quad Flat Package)、LQFP(Low rofile Quad Flat Package)，如图 2-2(b)所示；NXP 公司的 MCU，封装形式有 LQFP、TSOP(Thin Small Out-line Package)；Silicon Labs 公司的 MCU，封装形式有 TQFP(Thin Quad Flat Package)、QFN(Quad Flat No-lead Package)等。

对于初学者来说，DIP 封装的 MCU 使用方便，且便于记忆管脚位置，下面就结合图 2-2 介绍标准 89C51 DIP 封装形式的引脚功能。



(a) 40引脚双列直插(DIP)封装图　　(b) 44脚(QFP)封装图

图 2-2　标准 89C51 DIP 和 QFP 封装图

2.2.1　电源及时钟引脚

1. 电源引脚

MCU 使用的是+5 V 电源，其中 40 引脚(VDD)接正极，20 引脚(VSS)接地。

2. 时钟引脚

两个时钟引脚 XTAL1、XTAL2 外接晶体与片内的反相放大器构成一个振荡器，它为 MCU 提供了时钟信号。

2.2.2　并行 I/O 口

标准 89C51 MCU 具有 4 个 I/O 口，32 根 I/O 口线。

1. P0 口

P0 口是 8 位、漏极开路的双向 I/O 口。当扩展片外存储器(ROM 及 RAM)时，P0 口用作分时复用的地址/数据总线。在程序校验期间，输出指令字节(这时，需加外部上拉电阻)。P0 口(作为总线时)能驱动 8 个 LSTTL 负载。

2. P1 口

P1 口是 8 位、准双向 I/O 口，具有内部上拉电阻。在编程/校验期间，用作输入低位字节地址。P1 口可以驱动 4 个 LSTTL 负载。对于 89C52 来说，P1.0(T2)是定时器/计数器 T2 的计数输入端；P1.1(T2EX)是定时器/计数器 T2 捕获/重装载的外部触发输入端。这时，读两个特殊引脚的输出锁存器前，应由程序将相应输出锁存器置 1。

3. P2 口

P2 口是 8 位、准双向 I/O 口，具有内部上拉电阻。当扩展片外存储器(ROM 及 RAM)时，P2 口输出高 8 位地址。在编程/校验期间，接收高位字节地址。P2 口可以驱动 4 个 LSTTL 负载。

4. P3 口

P3 口是 8 位、准双向 I/O 口，具有内部上拉电阻。P3 口可以驱动 4 个 LSTTL 负载。P3 口还提供各种第二功能，在提供这些功能时，其输出锁存器应由程序置 1，其第二功能定义见表 2-1。

表 2-1　P3 口的第二功能定义

引　　脚	第二功能	说　　明
P3.0	RXD	串行数据输入口
P3.1	TXD	串行数据输出口
P3.2	$\overline{INT0}$	外部中断 0 输入口
P3.3	$\overline{INT1}$	外部中断 1 输入口
P3.4	T0	定时器 0 外部计数输入口
P3.5	T1	定时器 1 外部计数输入口
P3.6	$\overline{WR}$	外部数据存储器写选通输出口
P3.7	$\overline{RD}$	外部数据存储器读选通输出口

2.2.3　控制引脚

控制引脚共 4 根。

1. RST

RST 是复位输入信号端，高电平有效。振荡器工作时，在 RST 上作用两个机器周期以上的高电平，将使 MCU 复位。

2. $\overline{EA}$/VPP

$\overline{EA}$/VPP 是片外程序存储器访问允许信号，低电平有效。在编程时，其上施加 21V 或 12V 的编程电压。



3.　ALE/$\overline{\text{PROG}}$

ALE/$\overline{\text{PROG}}$ 输出地址锁存允许信号，当访问片外存储器时，锁存低字节地址。ALE 以 1/6 时钟频率的速率稳定输出信号，可用作对外输出的时钟或用于定时。在对程序存储器编程期间，用作输入，可输入编程脉冲($\overline{\text{PROG}}$)。ALE 可以驱动 8 个 LSTTL 负载。

4.　$\overline{\text{PSEN}}$

$\overline{\text{PSEN}}$ 是片外程序存储器选通信号，低电平有效。在从片外程序存储器取指令期间，在每个机器周期，当 $\overline{\text{PSEN}}$ 有效时，程序存储器的内容被送入 P0 口(数据总线)。$\overline{\text{PSEN}}$ 可以驱动 8 个 LSTTL 负载。

2.3　80C51 系列微控制器的 CPU 结构

80C51 系列 MCU 的 CPU 由运算器和控制器组成。

2.3.1　运算器

运算器主要用来对操作数进行算术运算、逻辑运算和位操作运算。它由算术逻辑运算单元 ALU(Arithmetic Logic Unit)、累加器 A(ACC，ACCumulator)、寄存器 B、暂存器、位处理器 CY、程序状态字 PSW 及 BCD 码修正电路等组成。

1.　算术逻辑运算单元 ALU

ALU 是 CPU 中必不可少的数据处理单元之一，功能十分强大。它可对 8 位变量进行逻辑与、逻辑或、逻辑异或、循环、求补和清零等基本操作，还可以进行加、减、乘、除等基本算术运算。此外，ALU 还具有位处理功能，可对位变量进行位处理，如置位、清零、求补、测试转移及逻辑与、逻辑或等操作。

2.　累加器 A

累加器 A 是 CPU 中使用最频繁的一个 8 位寄存器，简称 A 或 ACC 寄存器。

累加器 A 的作用如下。

(1) 累加器 A 是 ALU 单元的输入之一，同时它又是 ALU 运算结果的存放单元，即 ALU 运算结果又通过内部总线送入累加器 A 中存放。

(2) CPU 中的数据传送大多都通过累加器 A，故累加器 A 相当于一个数据的中转站。在 80C51 中，还有一部分可以不经过累加器的传送指令，如寄存器与直接寻址单元之间的数据传送，这样既加快了传送速度，又减少了累加器 A 的堵塞现象。

3.　寄存器 B

寄存器 B 是为乘法、除法操作而设置的寄存器，是 ALU 的输入之一。

乘法时 ALU 的两个输入分别为 A、B。运算结果即乘积的高 8 位放入 B 中，低 8 位放入 A 中；除法时，被除数放入 A 中，除数放入 B，商放在 A 中，余数放入 B 中。

在其他情况下，寄存器 B 可作为普通寄存器使用。

4. 程序状态字 PSW

程序状态字 PSW(Program Status Word)是一个逐位定义的 8 位的特殊功能寄存器，字节地址是 0D0H，它记录了程序运行状态的各种信息，有时，也称为程序状态寄存器。它是一个程序可访问的特殊功能寄存器，而且可以按位访问。PSW 的格式如图 2-3 所示。

MSB							LSB	
CY	AC	F0	RS1	RS0	OV	—	P	D0H
D7	D6	D5	D4	D3	D2	D1	D0	

图 2-3 PSW 的格式

其中 PSW.1 为 Intel 保留位，未用。其余 PSW 各位的功能如下。

(1) CY—PSW.7，进位标志位。它可被硬件或软件置位或清零。当两个 8 位数进行算术运算时高位发生了进位或借位都将 CY 置 1，不发生进位或借位时 CY 被清零。在位操作指令中它是位累加器。CY 也可写作 C。

(2) AC—PSW.6，辅助进位标志位。当进行 BCD 码的加法和减法时产生的低 4 位向高 4 位进位或借位时，AC 被硬件置 1，否则清零。

(3) F0—PSW.5，用户使用的标志位。可用软件将 F0 置 1 或清零，F0 一般用作控制程序流向的标志位。

(4) RS1、RS0—PSW.4、PSW.3，80C51 工作寄存器选择位。80C51 片内有 4 组工作寄存器组，可以用这两位来选择 4 组工作寄存器组中哪一组为当前工作寄存器组。

(5) OV—PSW.2，溢出标志位。当执行算术指令时，由硬件置 1 或清零，以指示运算是否产生溢出。

(6) P—PSW.0，奇偶标志位。它用来表示 A 中 8 位二进制数中 1 的个数是偶数还是奇数。若 P 为 1，则 A 中 1 的个数为奇数；若 P 为 0，则 A 中 1 的个数为偶数，即 80C51 系列 MCU 是一种偶校验 MCU。

2.3.2 控制器

控制器的作用是识别指令，并根据指令的性质控制 MCU 各功能部件。

控制器主要由程序计数器 PC(Program Counter)、程序地址寄存器、指令寄存器 IR、指令译码器、条件转移逻辑电路及时序控制逻辑电路构成。

1. 程序计数器 PC

程序计数器 PC 是一个独立的计数器，它存放着下一条将要从程序存储器中取出的指令的地址，我们一般将这个地址称为当前地址或 PC 的当前值。它的工作过程是：读指令时，程序计数器 PC 将其中的数作为所取指令的地址输出给程序存储器，然后程序存储器



按此地址输出指令字节，同时程序计数器 PC 本身自动加 1，指向下一条指令在程序存储器中的地址。

程序计数器 PC 中地址的变化决定着用户程序的流向。80C51 中程序计数器 PC 是一个 16 位的计数器，故其对程序存储器的寻址范围是 64 KB (2^{16}=65536=64 K)。

程序计数器 PC 的工作方式有以下几种。

(1) 程序计数器自动加 1，这是最基本的工作方式。

(2) 执行有条件或无条件转移指令时，程序计数器 PC 将被置入新的数值，从而使程序的流向发生变化。

(3) 在执行子程序调用或响应中断时，MCU 自动完成以下工作：将 PC 的当前值(此时，可称为断点值)自动送入堆栈；将子程序的入口地址或中断向量地址送入 PC，程序流向发生变化，执行子程序或中断服务程序。子程序或中断服务程序执行完毕，遇到返回指令 RET 或 RETI 时，将堆栈顶部的断点值弹出到 PC 中，程序的流向又返回到原来的地方，继续执行。

2. 指令寄存器 IR、指令译码器及定时控制逻辑电路

指令寄存器 IR 是用来专门存放指令操作码的专用寄存器。首先，从程序存储器中读出的指令先放入 IR 中；其次，将指令送入指令译码器，由指令译码器对该指令进行译码；最后，译码结果送定时控制逻辑，由定时控制逻辑发出一系列定时控制信号，控制 MCU 的各个功能部件进行相应的工作。对于运算指令还要把运算结果的特征送入程序状态寄存器 PSW 中。

2.4　80C51 系列微控制器的存储器结构

由于 80C51 系列 MCU 采用哈佛结构，程序存储器和数据存储器截然分开；片内集成存储器，而片外可扩展存储器；因此，80C51 系列 MCU 在物理上便有 4 个存储空间：片内程序存储器、片外程序存储器、片内数据存储器、片外数据存储器。存储器空间映像如图 2-4 所示。

1. 程序存储器空间

MCU 能够按照一定的次序工作，是由于程序存储器中存放了二进制代码形式的程序。程序存储器空间可划分为片内程序存储器空间和片外程序存储器空间两部分。

89C51 片内有 4KB 的程序存储器，可通过编程器对其编程，也可在线编程(ISP)。

如果 89C51 片内的 4KB 程序存储器不够用的话，80C51 系列 MCU 给用户提供了片外扩展至 64KB 程序存储器的能力。至于具体扩展多少程序存储器，用户需根据实际需要来决定。当然，由于 80C51 系列 MCU 技术的进步，片内程序存储器已可达 64KB。所以，用户也可根据需要，选择合适的片内程序存储器容量的 MCU。

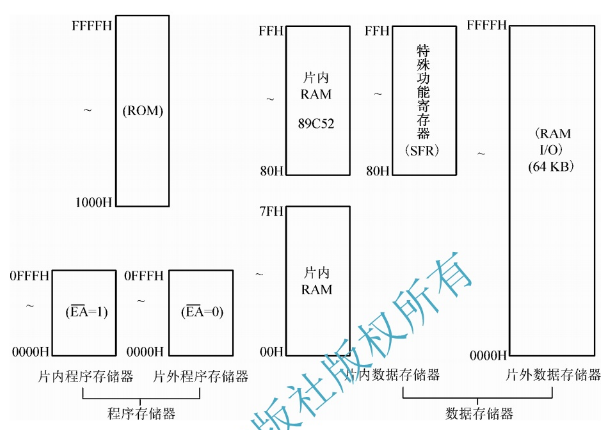


图 2-4 存储器空间映像图

2. 数据存储器空间

数据存储器空间用来储存程序在运行期间的变量和数据。数据存储器空间可分为片内数据存储器(Internal Data Memory，IRAM)和片外数据存储器(eXternal Data Memory，XRAM)两部分。

89C51 内部有 128B 的 RAM(89C52 为 256B)，其特点是存取速度快、功耗低。

当 89C51 内部 RAM 不够用时，用户可在片外扩展 RAM，最大可扩展至 64KB RAM，至于究竟扩展多少，应根据用户的实际需要决定。与程序存储器类似，在某些类型的 MCU 内部也集成了超过 256B 的 RAM，可根据实际需要选用，而不必再扩展 XRAM。需要注意的是，在片内集成的超过 256B 的 RAM，我们一般称作片内集成的 XRAM，需用 XRAM 的指令 MOVX 访问。

从图 2-4 中看出，片内数据存储器空间在物理上包含两部分：对于 89C51 来说，区间 00H～7FH 为 IRAM 区，区间 80H～0FFH 为 SFR 区；对于 89C52 来说，区间 00H～7FH 为 IRAM 区，区间 80H～0FFH 为 IRAM 区和 SFR 区的重叠区间。由于在 80H～0FFH 区间为重叠区间，所以，CPU 通过不同的寻址方式来访问 IRAM 区和 SFR 区。

在逻辑上，MCU 的存储器空间可划分为 3 个：片内、片外统一编址的 64KB 程序存储器空间；片内 256B(89C51)或 384B(89C52)数据存储器空间，其中包括特殊功能寄存器(Special Function Register，SFR)空间 128B；片外 64KB 的数据存储器空间，如图 2-4 所示。

由于地址重叠，在访问这 3 个不同的存储器空间时，应选用不同形式的指令。



2.4.1　程序存储器

程序存储器(ROM)用来存放 MCU 的系统程序、应用程序、数据或表格。首先，编译程序把用高级语言或汇编语言编写的程序变换成二进制数(bin)或十六进制数(hex)；其次，使用在系统可编程(ISP)方式或相应的编程器(Programmer)写入程序存储器中，用以控制 MCU 的工作。Intel 早期生产的 8031 内部无程序存储器，因此在使用这种 MCU 时必须在片外扩展程序存储器；89C51 内部有 4 KB 的程序存储器，可以用 ISP 方式或编程器给其写入程序，若其不够用时可进行外部扩展，外部扩展最多为 64 KB。

有关程序存储器的使用应注意以下几点。

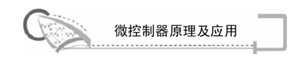
(1) $\overline{EA}$ 接高电平时程序将从片内程序存储器开始执行，当 PC 值超过片内 ROM 时自动转向片外程序存储器空间执行程序。$\overline{EA}$ 接低电平时 MCU 只能执行片外程序存储器中的程序。在现有技术下，我们只需要选用合适容量的 MCU 即可，不用再扩展程序存储器。因此，在设计电路时，$\overline{EA}$ 一般接高电平。

(2) 程序存储器的某些单元被固定用于中断源的中断服务程序的入口地址(中断向量地址或中断矢量地址)。80C51 系列 MCU 复位后，程序计数器的内容是 0000H，故所有的 MCU 系统必须从 0000H 单元开始取指令，执行程序。程序存储器中的 0000H 地址是系统程序的启动地址，或者称为复位或非屏蔽中断入口地址。这一点初学者一定要记牢。另外，对于 89C51 来说，还有 5 个单元具有特殊用途，它们是 5 个中断源的中断服务程序的入口地址；对于 89C52 来说，在 89C51 的基础上，再增加定时器/计数器 T2 的中断入口地址，见表 2-2。

表 2-2　中断入口地址表

中断源	入口地址
复位或非屏蔽中断	0000H
$\overline{INT0}$ 外部中断 0	0003H
定时器/计数器 0 中断	000BH
$\overline{INT1}$ 外部中断 1	0013H
定时器/计数器 1 中断	001BH
串行中断	0023H
定时器/计数器 2 中断(仅 89C52)	002BH

从表 2-2 可看出，程序启动地址至 $\overline{INT0}$ 入口地址仅有 3 个字节，所以，一般在这一地址存放一条绝对跳转指令，转向真正的用户主程序起始地址。从 $\overline{INT0}$ 入口地址开始，相邻中断入口地址仅间隔 8 个字节，一般也不足以存放中断服务程序，因此，通常在这些入口地址处也都有一条绝对跳转指令，转向真正的中断服务程序地址。



2.4.2 数据存储器

1. 片内数据存储器(IRAM)

片内数据存储器是最灵活的地址空间。它又分为三个部分：

(1) 片内低 128B 区，字节地址为 00H～7FH；

(2) 片内高 128B 区(89C51 没有，89C52 拥有)，字节地址为 80H～0FFH；

(3) 特殊功能寄存器(SFR)区，字节地址为 80H～0FFH。

图 2-5 为片内数据存储器各部分地址空间的分布图。

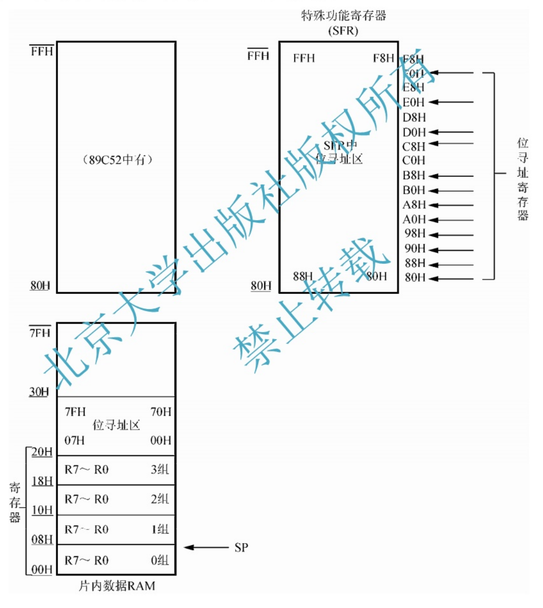


图 2-5　片内数据存储器的地址空间分布图

下面详细介绍片内数据存储器三部分的功能。

1) 片内低 128B 区

在片内低 128B 区，根据不同的寻址方式又可分为以下 3 个区域：工作寄存器区、位



寻址区、字节寻址区。

(1) 工作寄存器区。从图 2-5 可看出，字节地址为 00H～1FH 的 32 个单元是 4 组工作寄存器组，每个组含有 8 个字节的寄存器，其编号为 R0～R7，工作寄存器也称为通用寄存器。工作寄存器组是一个寄存器寻址的区域，指令均为单周期指令，数量最多，执行速度最快。

对这 4 组寄存器组来说，在同一时刻只能使用其中一组；使用哪一组作为当前工作寄存器组，是通过软件对程序状态寄存器(PSW)中的 RS0、RS1 位的设置来实现的(表 2-3)。这种配置方式给软件设计带来极大方便，特别是在中断嵌套时，实现工作寄存器现场保护极其方便。

表 2-3　工作寄存器组的选择

RS1	RS0	工作寄存器组(地址)
0	0	0 组(IRAM 地址 00H～07H)
0	1	1 组(IRAM 地址 08H～0FH)
1	0	2 组(IRAM 地址 10H～17H)
1	1	3 组(IRAM 地址 18H～1FH)

设置 RS0、RS1 的值时，可以采取对 PSW 进行字节寻址的方式，也可以采取对 RS0、RS1 进行直接位寻址的方式，间接或直接修改 RS0、RS1 的内容。通常采用位寻址比较方便。例如，系统复位时，RS0、RS1 的值均为 0，则选用工作寄存器组 0 为当前工作寄存器组。如果根据需要选用工作寄存器组 1，则只需将 RS0 的值改成 1 即可，可用位寻址方式(SETB RS0)来实现。

在 80C51 系列 MCU 的指令系统中，累加器 A，寄存器对 AB，数据指针 DPTR 及 PSW 中的进位位 CY 也是当作寄存器对待的，即对它们的寻址方式也是寄存器寻址方式。

寄存器 R0、R1 通常用作间接寻址时的地址指针。

(2) 位寻址区。从图 2-5 可看出，字节地址为 20H～2FH 的 16 个单元是可以位寻址的 IRAM 区。这 16 个字节单元，既可进行字节寻址，又可实现位寻址(共计 16×8 位=128 位)。字节地址与位地址之间的关系见表 2-4。这里要特别注意位地址和字节地址的区别，字节地址范围为 20H～2FH，位地址范围为 00H～7FH。位地址区的每一位都可以当作一个软件触发器使用，也可以表示数字电路中的一个逻辑变量。在编程当中，通常可以把各种程序状态标志、位控制变量存于位寻址区内。

表 2-4　RAM 中的位地址单元

字节地址	位地址							
	D7	D6	D5	D4	D3	D2	D1	D0
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	77H	76H	75H	74H	73H	72H	71H	70H


续表

字节地址	位地址							
	D7	D6	D5	D4	D3	D2	D1	D0
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

(3) 字节寻址区。从图 2-5 可看出，地址为 30H～7FH 的存储单元为普通 IRAM 区，只能进行字节寻址，可以采用直接寻址方式来访问每一个字节。这一区域，一般称为用户 RAM 区。

2) 片内高 128B 区

从图 2-5 可看出，对于 89C52 MCU 来说，高 128B 的 IRAM 区和 SFR 区的地址空间是重叠的。究竟访问哪一个区通过不同的寻址方式来加以区别，访问高 128B IRAM 区时，选用间接寻址方式；访问 SFR 区时，则应选用直接寻址方式。对于 89C52 MCU 来说，用户 RAM 区还包括 80H～0FFH 的存储单元。因此，89C52 MCU 用户区包含 30H～0FFH，共 208 个存储单元。

3) 特殊功能寄存器(SFR)区

特殊功能寄存器 SFR(Special Function Register)是 80C51 系列 MCU 中各功能部件所对应的寄存器，是用来存放相应功能部件的控制命令、状态或数据的区域。这是 80C51 系列 MCU 中最有特色的部分。现在所有 80C51 系列 MCU 功能的增加和扩展几乎都是通过增加特殊功能寄存器来实现的。

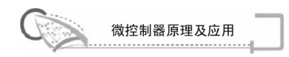
80C51 系列 MCU 设有 128B 片内数据 RAM 结构的特殊功能寄存器空间区域。除程序计数器 PC 和 4 个通用工作寄存器组外，其余所有的寄存器都安排在这个地址空间之内。



对于 89C51 MCU 来说，共定义了 21 个特殊功能寄存器，其名称和字节地址列于表 2-5 中。而在 89C52 MCU 中，除上述的 21 个特殊功能寄存器之外，还增加了 5 个特殊功能寄存器，共计 26 个，见表 2-5。

表 2-5　SFR 的名称及其分布

标识符	名　称	字节地址	位地址
B	B 寄存器	0F0H	0F7H～0F0H
A(或 ACC)	累加器	0E0H	0E7H～0E0H
PSW	程序状态字	0D0H	0D7H～0D0H
TH2	定时器/计数器 2 高字节	0CDH	—
TL2	定时器/计数器 2 低字节	0CCH	—
RCAP2H	定时器/计数器 2 捕获寄存器高字节	0CBH	—
RCAP2L	定时器/计数器 2 捕获寄存器低字节	0CAH	—
T2CON	定时器/计数器 2 控制寄存器	0C8H	0CFH～0C8H
IP	中断优先级控制寄存器	0B8H	0BFH～0B8H
P3	P3 口	0B0H	0B7H～0B0H
IE	中断允许控制寄存器	0A8H	0AFH～0A8H
P2	P2 口	0A0H	0A7H～0A0H
SBUF	串行数据缓冲器	99H	—
SCON	串行控制寄存器	98H	9FH～98H
P1	P1 口	90H	97H～90H
TH1	定时器/计数器 1 高字节	8DH	—
TH0	定时器/计数器 0 高字节	8CH	—
TL1	定时器/计数器 1 低字节	8BH	—
TL0	定时器/计数器 0 低字节	8AH	—
TMOD	定时器/计数器方式控制器	89H	—
TCON	定时器/计数器控制寄存器	88H	8FH～88H
PCON	电源控制寄存器	87H	—
DPH	16 位数据指针高字节	83H	—
DPL	16 位数据指针低字节	82H	—
SP	堆栈指针	81H	—
P0	P0 口	80H	87H～80H



从表 2-5 中可以发现一个规律：8 位地址的末位为 0H 或 8H 的特殊功能寄存器不但可以字节寻址，还可以进行位寻址。这些寄存器与片内数据存储器地址为 20H～2FH 的 16 个字节共同构成了可位寻址的空间，可以使用位操作指令逐位访问。这些特殊功能寄存器与位地址的关系见表 2-6。从表 2-6 中可看出，有些可以位寻址的位还有其特殊的名称。

表 2-6　SFR 与位地址的对应关系

SFR		位地址							
名称	字节地址	D7	D6	D5	D4	D3	D2	D1	D0
B	0F0H	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H
ACC	0E0H	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H
PSW	0D0H	CY	AC	F0	RS1	RS0	OV	—	P
		D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
T2CON	0C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{\mathrm{T}}$2	CP/$\overline{\mathrm{RL}}$2
		CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H
IP	0B8H	—	—	PT2	PS	PT1	PX1	PT0	PX0
		AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
P3	0B0H	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H
IE	0A8H	EA	—	ET2	ES	ET1	EX1	ET0	EX0
		AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
P2	0A0H	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H
SCON	98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
		9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
P1	90H	97H	96H	95H	94H	93H	92H	91H	90H
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
		8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
P0	80H	87H	86H	85H	84H	83H	82H	81H	80H

特殊功能寄存器在存储空间中的分布见表 2-7。从表 2-7 中可看出，在 128B 的空间中存在大量空白区，这为 80C51 系列 MCU 的功能增加提供了极大可能性。需要注意的是，除了所定义的特殊功能寄存器外，读写 SFR 区其他未定义的存储单元，将会得到一个不确定的随机数(对于 Silicon Labs 公司来说，由于扩展的功能太多，128B 空间也不够用，它采用页的方式增加特殊功能寄存器)。



表 2-7　SFR 在存储空间中的分布

低位地址	高位地址							
	8	9	A	B	C	D	E	F
0	P0	P1	P2	P3	—	PSW	ACC	B
1	SP	—	—	—	—	—	—	—
2	DPL	—	—	—	—	—	—	—
3	DPH	—	—	—	—	—	—	—
4	—	—	—	—	—	—	—	—
5	—	—	—	—	—	—	—	—
6	—	—	—	—	—	—	—	—
7	—	PCON	—	—	—	—	—	—
8	TCON	SCON	IE	IP	T2CON	—	—	—
9	TMOD	SBUF	—	—	—	—	—	—
A	TL0	—	—	—	RCAP2L	—	—	—
B	TL1	—	—	—	RCAP2H	—	—	—
C	TH0	—	—	—	TL2	—	—	—
D	TH1	—	—	—	TH2	—	—	—
E	—	—	—	—	—	—	—	—
F	—	—	—	—	—	—	—	—

特殊功能寄存器中的累加器 A、程序状态字(PSW)、寄存器 B 已经在前面介绍过，下面介绍几个常用的特殊功能寄存器，其余特殊功能寄存器将在后续相关章节中介绍。

(1) 堆栈指针 SP。堆栈(Stack)是一块按照“先进后出，后进先出”原则组织的存储器空间，有时也简称为栈。一般来说，大部分的 CPU 都将堆栈设置在 RAM 区，80C51 系列 MCU 也不例外。

80C51 系列 MCU 的堆栈指针 SP 是一个 8 位的 SFR，SP 的内容指示出堆栈顶部(简称栈顶)在 IRAM 中的位置，且 SP 指向的总是最后压入堆栈的 8 位数据。例如，80C51 系列 MCU 复位后 SP 的内容为 07H，即指向 07H 的 RAM 单元，因此，新的数据将从 08H 开始压入堆栈。考虑到 08H～1FH 为工作寄存器区，20H～2FH 为位寻址区，因此，在设计程序时要把 SP 值改为大于 2FH 的值，即将堆栈设置在字节寻址区(用户 RAM 区)。对于 80C51 系列 MCU 来说，对堆栈的操作有两种方式：一种是数据压入堆栈(简称入栈，指令为 PUSH)；另一种是数据弹出堆栈(简称出栈，指令为 POP)。当进行入栈操作时，首先将 SP 的内容自动加 1，指向新的存储单元，再把数据压入存储单元；当进行出栈操作时，首先将当前栈顶的内容弹出到相应位置(由指令 POP 确定)，然后把 SP 的内容自动减 1。

堆栈的作用主要是为子程序调用或中断操作而设置的。它的功能有两个：保护断点和保护现场。在进入子程序和中断服务程序前，CPU 会将断点地址存入堆栈，返回主程序时

恢复，这就是所谓的保护断点。在 CPU 执行子程序或中断服务程序后要用到 RAM 中的一些存储单元，如果在主程序中恰好也用到这些存储单元，这样就会破坏这些存储单元中的原有内容。为了能在子程序或中断服务程序中使用 RAM 中的这些存储单元，要在转去执行子程序和中断服务程序之前把 RAM 中的有关存储单元的内容保存起来，返回主程序后恢复，这就是所谓的现场保护。此外，堆栈也用于数据的临时存放。

(2) 数据指针 DPTR。数据指针 DPTR(Data PoinTeR)是一个 16 位的 SFR，它的高位字节为 DPH，低位字节为 DPL。它可以作为一个 16 位的寄存器 DPTR 使用，也可以作为两个 8 位的寄存器 DPH 和 DPL 来使用。80C51 系列 MCU 对外部 RAM 进行操作时要利用 DPTR 作为数据指针，即 DPTR 存放外部 RAM 的 16 位地址，通过它来访问 64KB 的片外 RAM 或片外 ROM。

2. 片外数据存储器(XRAM)

片外数据存储器是在外部存放数据的区域，这一区域用寄存器间接寻址的方法访问，所用的寄存器为 DPTR、R1 或 R0。当用 R1、R0 寻址时，由于 R0、R1 为 8 位寄存器，因此最大寻址范围为 256B；当用 DPTR 寻址时，由于 DPTR 为 16 位寄存器，因此最大寻址范围为 64 KB。

2.5 时钟电路与 CPU 的工作时序

时钟电路用于产生 MCU 工作所需要的时钟信号，而时序所研究的是指令执行中各信号之间的相互关系。MCU 本身就如一个复杂的同步时序电路，为了保证同步工作方式的实现，电路应在唯一的时钟信号控制下严格地按时序进行工作。

2.5.1 时钟电路

MCU 工作时，是在统一的时钟脉冲控制下有序进行的。这个脉冲是由时钟电路(图 2-6)产生的。时钟电路由振荡器和分频器组成，振荡器产生基本的振荡信号，然后进行分频，得到相应的时钟。振荡电路有两种振荡方式：内部振荡和外部振荡。

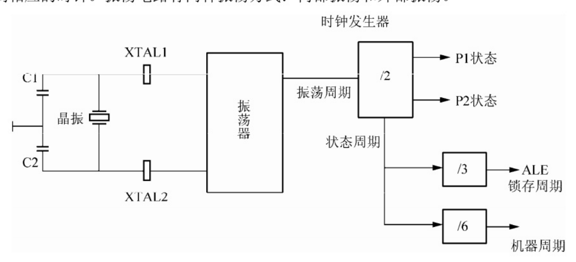


图 2-6 时钟电路



1. 内部振荡方式

80C51 系列 MCU 内有一个用于构成振荡器的高增益反相放大器，引脚 XTAL1 和 XTAL2 分别是此放大器的输入端和输出端。把放大器与作为反馈元器件的晶体振荡器和陶瓷电容相连，就构成了自激振荡器，其输出就是时钟脉冲。内部振荡电路如图 2-7 所示。

2. 外部振荡方式

外部振荡方式是把外部已有的时钟信号引入 MCU 内部。对于 HMOS 型 MCU，外部振荡电路如图 2-8 所示。对于 CHMOS 型 MCU，XTAL1 接片外振荡脉冲输入端，XTAL2 悬空。

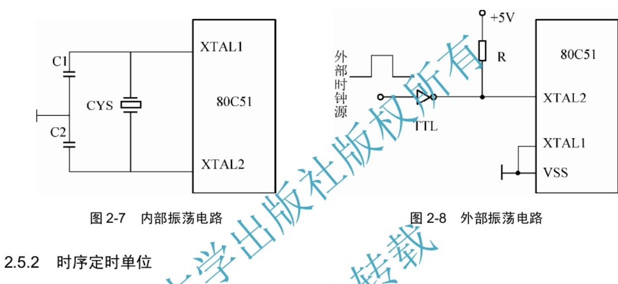


图 2-7　内部振荡电路　　　图 2-8　外部振荡电路

2.5.2　时序定时单位

MCU 执行指令是在时钟脉冲控制下一步一步进行的，由于指令的功能和长短各不相同，因此，指令执行所需的时间也不一样。80C51 系列 MCU 的时序定时单位共有 4 个，从小到大依次是振荡周期、状态周期、机器周期和指令周期。

(1) 振荡周期：晶体振荡器输出的时钟周期。

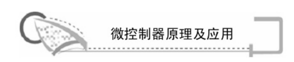
(2) 状态周期：振荡信号经二分频后形成的时钟脉冲信号，用 S 表示。一个状态周期的两个振荡周期作为两个节拍分别称为节拍 P1 和节拍 P2。在 P1 有效时，通常完成算术逻辑操作；在 P2 有效时，一般进行内部寄存器之间的传输。

(3) 机器周期：通常将完成一个基本操作所需要的时间称为机器周期。80C51 系列 MCU 的一个机器周期包括 6 个状态周期，用 S1，S2，…，S6 表示；共 12 个节拍，依次可表示为 S1P1，S1P2，S2P1，S2P2，…，S6P1，S6P2。

(4) 指令周期：CPU 执行一条指令所需要的时间为一个指令周期。显然，指令不同，对应的指令周期也不一样。一个指令周期通常含有 1～4 个机器周期。80C51 系列 MCU 除了乘法、除法指令是 4 个机器周期外，其余都是单周期指令或双周期指令。

80C51 系列 MCU 的典型取指令、执行指令的时序如图 2-9 所示。

由图 2-9 可知，在每一个机器周期内，地址锁存信号 ALE 出现两次有效信号，即两次高电平信号。第一次出现在 S1P2 和 S2P1 期间，第二次出现在 S4P2 和 S5P1 期间。



对于单字节单周期指令，在一个机器周期内，读两次操作码，但在 S4P2 期间所读的这个字节操作码被丢掉。

如果是双字节单周期指令，则在 S4 期间读指令的第二个字节。

对于单字节双周期指令，两个机器周期内发生 4 次读操作码的操作，由于是单字节指令，后 3 次读操作都无效。

在图 2-9 中只表示了取指令操作的有关时序，而没有说明执行指令的时序。实际上每条指令都有具体的数据操作，如算术运算和逻辑操作一般发生在 P1 期间，片内存储器之间的数据传送操作发生在 P2 期间等。有兴趣的读者可以参阅相关资料，此处不再赘述。

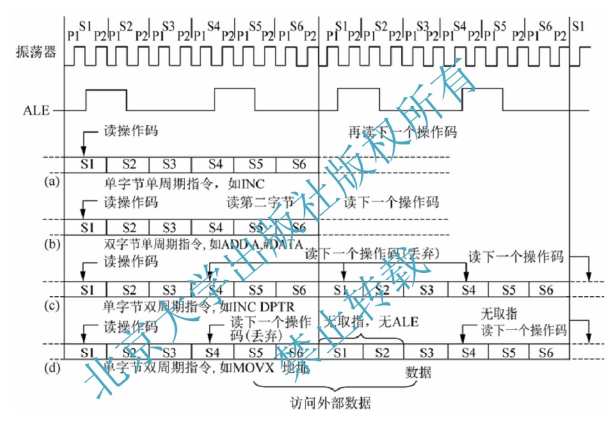


图 2-9　80C51 系列 MCU 取指令、执行指令的时序图

2.6　80C51 系列微控制器的工作方式

80C51 系列 MCU 共有复位、程序执行、低功耗及编程四种工作方式。

2.6.1　复位方式

1. 复位操作

复位是 MCU 的初始化操作，只要给 RST 引脚加上 2 个机器周期以上的高电平信号即可实现复位操作。复位的主要功能是把程序计数器 PC 的值初始化为 0000H，使 MCU 从 0000H 单元开始执行程序。任何一个 MCU 系统都要设置一个复位按键，当程序运行出错或操作错误使系统处于死机状态时，用它来摆脱死机状态。复位的另外一个功能可以使其



内部的寄存器处于复位状态。89C51 内部寄存器的复位状态见表 2-8。

表 2-8　89C51 内部寄存器的复位状态

寄存器	复位状态	寄存器	复位状态
PC	0000H	TMOD	00H
ACC	00H	TCON	00H
PSW	00H	TH0	00H
B	00H	TL0	00H
SP	07H	TH1	00H
DPTR	0000H	TL1	00H
P0～P3	FFH	SCON	00H
IP	×××00000B	SBUF	××××××××B
IE	0××00000B	PCON	0×××0000B

另外在复位有效期间(即高电平持续期间)，MCU 的 ALE 和 $\overline{\text{PSEN}}$ 引脚均为高电平，其内部的 RAM 不受复位的影响。

2. 复位电路

80C51 系列 MCU 的复位是由外部的复位电路来实现的。一般的复位电路如图 2-10 所示。

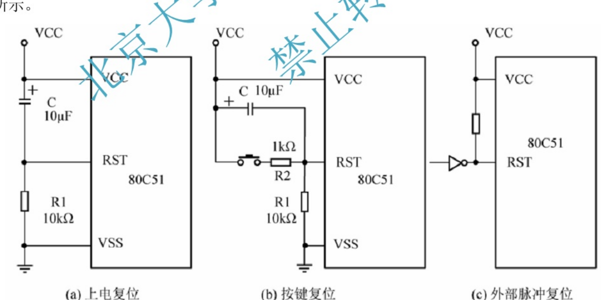


图 2-10　复位电路

图 2-10(a)所示为上电复位电路，利用电容充放电来实现复位，当加电时，电容电压不能突变，RST 引脚为高电平，开始复位；随着电容两端电压的升高，电阻两端的电压不断下降，当电容充满电后，电阻相当于下拉电阻，使 RST 一直处于低电平，复位结束。复位

的时间与充电的时间有关，充电时间越长复位时间越长，增大电容和电阻可以增加复位时间。

图 2-10(b)所示为按键复位电路，当按下按键时，通过两个电阻分压，使 RST 引脚产生高电平，复位时间由按键按下的时间决定。

图 2-10(c)所示为外部脉冲复位电路。当复位低电平加到反相器输入端且保持时间大于 2 个机器周期时，将复位 CPU。

2.6.2 程序执行方式

程序执行方式是 MCU 最基本的工作方式。由于复位后 PC=0000H，因此程序执行总是从地址 0000H 开始的。但一般情况下，程序并不是真正从 0000H 开始(例如，中断入口地址显然不能存放主程序，主程序的入口地址应该跳过这一区域)，为此就必须在 0000H 开始的单元中存放一条无条件转移指令，以便跳转到实际主程序的入口去执行。

2.6.3 低功耗方式

80C5l 系列 MCU 有两种低功耗方式，即空闲方式(Idle Mode)和掉电方式(Power Down Mode)。空闲方式和掉电方式都是由电源控制寄存器(PCON)的有关位来控制的。电源控制寄存器是一个逐位定义的 8 位寄存器，其格式如图 2-11 所示。

MSB							LSB	
SMOD	—	—	—	GF1	GF0	PD	IDL	87H
D7	D6	D5	D4	D3	D2	D1	D0	

图 2-11 PCON 的格式

下面对 PCON 的各位加以说明。

(1) SMOD——波特率倍增位，在串行通信时使用。

(2) GF1——通用标志位 1。

(3) GF0——通用标志位 0。

(4) PD——掉电方式位，PD=1，则进入掉电方式。

(5) IDL——空闲方式位，IDL=1，则进入空闲方式。

要想使 MCU 进入空闲方式或掉电方式，只要执行一条对 IDL 位或 PD 位置 1 的指令即可。

1. 空闲方式

如果使用指令使 PCON 寄存器 IDL 位置 1，则 MCU 进入空闲方式。这时振荡器仍然运行，并向中断逻辑、串行口和定时器/计数器电路提供时钟，但向 CPU 提供时钟的电路被阻断，因此 CPU 不能工作，而中断功能继续存在，但与 CPU 有关的，如 SP、PC、PSW、ACC 及全部通用寄存器都被“冻结”在原状态。

在空闲方式下，若引入一个外中断请求信号，在 MCU 响应中断的同时，IDL 位被硬件自动清零，MCU 就退出空闲方式而进入正常工作方式。中断服务程序结束后，通过 RETI 指令，就可以使 MCU 恢复正常工作后，返回断点继续执行程序。



2. 掉电方式

PD 位控制 MCU 进入掉电方式。当把 PD 位置 1 时，MCU 便进入掉电方式。此时 MCU 的振荡器停止工作，芯片停止所有功能，但片内 RAM 和 SFR 内容保持不变。

80C51 系列 MCU 退出掉电方式的唯一方法是硬件复位。只要硬件复位信号出现，就能使 MCU 退出掉电方式。

2.6.4　编程方式

80C51 系列 MCU 一般具有两种编程方式：并行编程方式和串行编程方式。对于并行编程方式，一般使用编程器(Programmer)将程序写入 MCU。对于串行编程方式，各公司有不同的编程方法及接口，但它们都支持在系统编程(In System Programming，ISP)。例如，Atmel 公司 89S51 系列 MCU 使用 SPI 接口将程序写入 Flash ROM；STC 公司及 NXP 公司的 MCU 则使用 UART 接口将程序写入 Flash ROM；而 Silicon Labs 公司则使用 JTAG 接口或 C2 接口将程序写入 Flash ROM。STC 公司 MCU 已经固化有 ISP 引导码，并设置为上电复位进入 ISP。下面我们以 STC 公司 MCU 为例介绍串行编程方式。

STC 公司的 MCU 在线编程典型电路如图 2-12 所示。从图中可看出，要实现 ISP，需要电平转换电路 RS232(具体芯片可选用 STC232/MAX232/SP232，其工作电压为 4.5～5.5V；或 STC3232/MAX3232/SP3232 其工作电压为 3～5.5V)，计算机 9 芯串口，以及必要的无源器件。

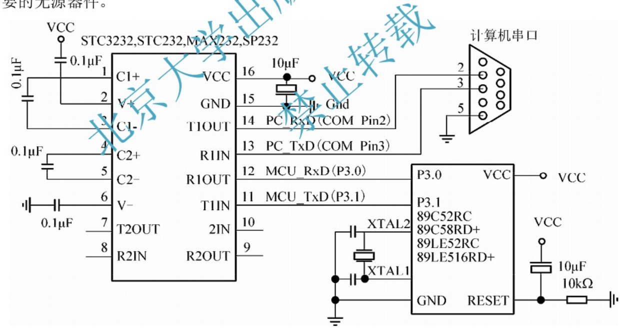


图 2-12　STC 公司的 MCU 在线编程典型电路

STC 公司的 MCU 串行编程步骤如下。

(1) 下载并安装 STC 公司提供的 ISP 工具 STC-ISP.EXE。

(2) 使用串行线将计算机的串口与编程电路的串口连接起来。

(3) 运行 STC-ISP，编程界面如图 2-13 所示。

图 2-13　STC-ISP 编程界面

(4) STC-ISP 设置步骤如下。

Step1/步骤 1：选择你所使用的 MCU 型号，如 STC89C52RC、STC89LE516AD 等。

Step2/步骤 2：按下按钮“打开程序文件”以打开文件。

Step3/步骤 3：选择串行口，如串行口 1—COM1，串行口 2—COM2，…。有些新式笔记本电脑没有 RS232 串行口，可买一条 USB-RS232 转接器。

Step4/步骤 4：Double speed/双倍速设置。设置是否双倍速，一般标准 89C51 时钟选择 12T/单倍速；如需提高 MCU 的工作速度，可选择 6T/双倍速。STC89C51RC/RD+系列可以反复设置双倍速/单倍速，新的设置停电后重新冷启动后才能生效。

Step5/步骤 5：选择“Download/下载”按钮将用户的程序写入 MCU 内部，可重复执行 Step5/步骤 5，也可选择“Re-Download/重复下载”。按钮下载时注意看提示，主要看是否要给 MCU 上电或复位，下载速度比一般通用编程器快。一般先选择按下“Download/下载”按钮，然后再给 MCU 上电复位(即按下“Download/下载”按钮前，电源是断开的)。

所有步骤中，除以上选项外，其余选项可使用默认状态。但需要特别注意的是，如果在 Step5/步骤 5 中选项“当目标代码发生变化后自动调入文件，并立即发送下载命令”处于未选中状态(软件默认状态)时，源程序一旦发生变化并再次编译，就需要再次执行 Step2/步骤 2。



本 章 小 结

1. 80C51 系列微控制器的硬件组成

MCU 的结构特征是将组成计算机的基本部件集成在一块电路芯片上，有的 MCU 生产厂家还集成了一些应用于测量和控制的特殊部件，从而构成了一片强大的计算机系统芯片，主要包括以下几部分：微处理器(CPU)、程序存储器(ROM)、数据存储器(RAM)、可编程并行 I/O 口、全双工串行口(UART)、定时器/计数器(T/C)、中断控制系统、定时电路等。

2. 89C51 微控制器的引脚介绍

89C51 芯片是 DIP(Dual In-line Package)封装的，有 40 个引脚，如图 2-2 所示。89C52 芯片与 89C51 芯片封装形式一样。由于技术的进步以及可靠性、小型化的要求，在应用中，现在的 MCU 出现了多种封装形式。图 2-2 是标准 89C51DIP 封装图。

(1) 电源及时钟引脚：MCU 使用的是+5 V 电源，其中 40 引脚(VDD)接正极，20 引脚(VSS)接地；两个时钟引脚 XTAL1、XTAL2 外接晶体与片内的反相放大器构成一个振荡器，它为 MCU 提供了时钟信号。

(2) 并行 I/O 口：标准 89C51 MCU 具有 4 个 I/O 口(P0、P1、P2、P3 口)，32 根 I/O 口线。

(3) 控制引脚：控制引脚共 4 根：RST、$\overline{EA}$/VPP、ALE/$\overline{PROG}$、$\overline{PSEN}$。

3. 80C51 系列微控制器 CPU 的结构

80C51 系列 MCU 的 CPU 由运算器和控制器组成。运算器主要用来对操作数进行算术运算、逻辑运算和位操作运算；控制器的作用是识别指令，并根据指令的性质控制 MCU 各功能部件。

4. 80C51 系列微控制器的存储器结构

由于 80C51 系列 MCU 采用哈佛结构，程序存储器和数据存储器截然分开；片内集成存储器，而片外可扩展存储器；因此，80C51 系列 MCU 在物理上便有 4 个存储空间：片内程序存储器、片外程序存储器、片内数据存储器、片外数据存储器。

5. 时钟电路与 CPU 的工作时序

时钟电路用于产生 MCU 工作所需要的时钟信号，而时序所研究的是指令执行中各信号之间的相互关系。MCU 本身就如一个复杂的同步时序电路，为了保证同步工作方式的实现，电路应在唯一的时钟信号控制下严格地按时序进行工作。

(1) 时钟电路：MCU 工作时，是在统一的时钟脉冲控制下有序进行的。这个脉冲是由时钟电路产生的。时钟电路由振荡器和分频器组成，振荡器产生基本的振荡信号，然后进行分频，得到相应的时钟。振荡电路有两种方式：内部振荡和外部振荡，如图 2-7 和图 2-8 所示。

(2) 工作时序：MCU 执行指令是在时钟脉冲控制下一步一步进行的，由于指令的功能和长短各不相同，因此，指令执行所需的时间也不一样。80C51 系列 MCU 的时序定时单位共有 4 个，从小到大依次是振荡周期、状态周期、机器周期和指令周期。80C51 系列

MCU 的典型取指令、执行指令的时序如图 2-9 所示。

6. 80C51 系列微控制器的工作方式

80C51 系列 MCU 共有复位、程序执行、低功耗以及编程 4 种工作方式。

(1) 复位方式：复位是 MCU 的初始化操作，只要给 RST 引脚(图 2-2)上加上 2 个机器周期以上的高电平信号即可实现复位操作。复位的主要功能是把程序计数器 PC 的值初始化为 0000H，使 MCU 从 0000H 单元开始执行程序。80C51 系列 MCU 的复位是由外部的复位电路来实现的。一般的复位电路如图 2-10 所示，这种复位电路属于电平复位电路。

(2) 程序执行方式：程序执行方式是 MCU 的最基本工作方式。由于复位后 PC=0000H，因此程序执行总是从地址 0000H 开始的。但一般情况下，程序并不是真正从 0000H 开始，为此就必须在 0000H 开始的单元中存放一条无条件转移指令，以便跳转到实际主程序的入口去执行。

(3) 低功耗方式：80C51 系列 MCU 有两种低功耗方式，即空闲方式(Idle Mode)和掉电方式(Power Down Mode)。空闲方式和掉电方式都是由电源控制寄存器(PCON)的有关位来控制的。电源控制寄存器是一个逐位定义的 8 位寄存器，其格式如图 2-11 所示。

(4) 编程方式：80C51 系列 MCU 一般具有两种编程方式：并行编程方式和串行编程方式。对于并行编程方式，一般使用编程器(Programmer)将程序写入 MCU；对于串行编程方式，各公司有不同的编程方法及接口，但它们都支持 ISP(In System Programming，在系统编程)。

思考题与习题

1. 80C51 系列 MCU 的哪些芯片引脚具有第二功能？各功能是什么？

2. 80C51 系列 MCU 的存储器在结构上有何特点？在物理上有哪几种空间？在逻辑上有哪几种空间？访问片内 RAM 和片外 RAM 的指令有何区别？

3. 80C51 系列 MCU 的片内 RAM 低 128 字节划分为哪几个部分？各部分主要功能是什么？

4. 80C51 系列 MCU 设有几个通用工作寄存器组？有什么特点？如何选用？

5. 什么是现场保护？如何实现工作寄存器组的现场保护？

6. 堆栈的功能是什么？堆栈指针(SP)的作用是什么？在程序设计时，为什么要修改 SP 的值？

7. 请写出 80C51 系列 MCU 的中断入口地址。

8. 请简述 80C51 系列 MCU 的时钟周期(振荡周期)、状态周期、机器周期、指令周期的概念及其关系。

9. 80C51 系列 MCU 的复位有哪几种方法？复位后 MCU 的各寄存器及 RAM 的初始状态如何？

第 3 章

80C51 系列微控制器的指令系统及程序设计

本章教学要点

知识要点	掌握程度	相关知识
微控制器概述	了解微控制器指令分类、格式； 熟悉微控制器各种指令符号	微控制器的指令分类； 格式及其符号
寻址方式	熟悉微控制器寻址方式； 掌握各种寻址方式的表达	立即寻址；直接寻址；寄存器寻址；寄存器间接寻址；变址寻址；相对寻址；位寻址
指令系统	掌握各种指令的传递方式	数据传送类；算术运算类；逻辑运算类；控制转移类；位操作类
汇编语言程序	熟悉汇编语言； 掌握汇编语言程序设计	汇编语言语句和格式； 汇编方式； 汇编语言程序设计



导入案例

微控制器的大脑——指令系统

——《机械公敌》剧情



公元 2035 年，智能型机器人已被人类广泛利用。作为最好的生产工具和人类伙伴，机器人在各个领域扮演着日益重要的角色，而由于众所周知的机器人“三大安全法则”的限制，人类对这些能够胜任各种工作且毫无怨言的伙伴充满信任，它们中的很多甚至已经成为一个家庭成员。

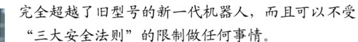
总部位于芝加哥的 USR 公司开发出更先进的 NS-5 型超能机器人，并计划达到平均每 5 人便可拥有 1 个这种机器人。然而就在新产品上市前夕，机器人的创造者阿尔弗莱德·朗宁博士却在公司内离奇死亡。

怀念以往简单生活，爱听老歌，喜欢老式打扮的黑人警探戴尔·史普纳接受了此案的调查工作。他根据对朗宁博士生前在 3D 投影机内留下的信息进行分析和对自杀现场的勘查，怀疑这起案件并非人类所为，而公司总裁劳伦斯·罗伯逊似乎也与此事有关。调查过程中他遇到了专门从事机器人心理研究的科学家苏珊·卡尔文博士，希望在她的帮助下找到答案。向来崇尚逻辑与科学的苏珊坚信，机器人不仅会最大限度地帮助人类进步，而且绝不会违背“三大安全法则”而对人类有所伤害。过去的经历令史普纳对机器人深感厌恶，他对于机器人安全性的怀疑与苏珊的坚定南辕北辙。

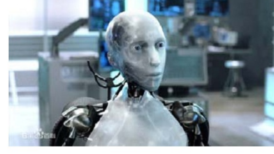


史普纳发现了一个名叫桑尼的机器人极有可能就是奉命杀害朗宁博士的“凶手”。在追捕中他发现桑尼不仅具有自我思考能力，而且拥有酷似人类的情感。讯问中桑尼告诉史普纳，他并没有杀害朗宁博士，而是在帮助他做一件事情。劳伦斯以只有人杀人才能定罪，机器人杀人只能认定为“工业意外”为由将桑尼带回公司。这给调查带来了极大的困难，却更坚定了史普纳追查到底的决心。

史普纳继续追踪一切与朗宁博士有关的资料。调查使他遭到大批 NS-5 型机器人的追杀，显然是有人想置史普纳于死地。苏珊来到史普纳家中，告知在对桑尼的检查中发现，他不仅是



完全超越了旧型号的新一代机器人，而且可以不受“三大安全法则”的限制做任何事情。



苏珊无意中惊讶地发现，史普纳原来是一个利用高科技修复合成的人。史普纳向苏珊讲述了他几年前经历的一场严重车祸。前来救助的机器人以存活率为依据，放弃了一个 12 岁女孩的生命，这就是他对机器人冷酷无情、深感厌恶的原因。劳伦斯承



认他知道确实存在着能够超越“三大安全法则”的机器人，而自己也在尽力挽回这个由朗宁博士犯下的错误，并要求苏珊尽快将桑尼销毁。桑尼将自己的梦画在纸上交给了史普纳，希望这个被朗宁博士载入的信息能够对他有所帮助。在执行销毁命令时，苏珊使用调包计将桑尼救出。

根据桑尼梦中的情景，史普纳找到一个机器人的存放基地。他发现一项所谓的“人类保护计划”正在实施。新一代 NS-5 型机器人正在基地奉命销毁所有旧型号机器人，然而这仅仅是计划的第一步。与此同时，大批新型机器人走上街头命令市民回到家中，并对抗议的市民实行宵禁。机器人与人类之间发生了激烈的冲突，此时机器人已完全不受人类操纵，整个城市顷刻间被它们控制。苏珊也被自家的机器人限制了自由，幸好史普纳及时赶到将她救出。

史普纳、苏珊和桑尼一起来到 USR 公司总部，却发现劳伦斯已死。史普纳忽然意识到自己的错误，应该怀疑的其实并非人类。真正的幕后操纵者竟然是公司名为“薇琪”的中央控制系统。正是“她”利用上层控制系统囚禁了朗宁博士并对机器人进行操控。“薇琪”的影像出现在他们面前。“她”认为人类正在危害自身的安全，国家发动战争，人类摧残地球，而机器人必须拯救人类，保证人类的持续存在发展，因此利用了上行线路控制了 NS-5 型机器人的程序来实施拯救计划。

真相大白，而此时唯一能做的就是尽快用抹除剂摧毁“薇琪”的智能控制系统，制止这场人类的灾难。大批受到控制的机器人向他们涌来。双方经过激烈的战斗，纳米机器人终于注入“薇琪”的智能系统，“人类保护计划”的所有命令随即终止。城市恢复了正常，机器人重新开始为人类服务。史普纳也终于消除了对机器人的怀疑和厌恶，与桑尼成为好朋友。

从《机械公敌》这部电影可以看出，机器人是正义的还是邪恶的，不取决于机器人的硬件组成，而是取决于机器人的智能控制系统。智能控制系统是由程序组成的，而程序最终是由 CPU 的指令组成的。

CPU 作为一台计算机的核心，它的作用是无法替代的。而 CPU 本身只是在块硅晶片上所集成的超大规模的集成电路，集成的晶体管数量可达到上亿个，是由非常先进复杂的制造工艺制造出来的，拥有相当高的科技含量。一颗如此精密的芯片是如何控制一个庞大而复杂的电脑系统呢？就是靠 CPU 中所集成的指令系统。所谓指令系统，就是 CPU 中用来计算和控制计算机系统的一套指令的集合，而每一种新型的 CPU 在设计时就规定了一系列与其他硬件电路相配合的指令系统。而指令系统的先进与否，也关系到 CPU 的性能发挥，它也是 CPU 性能体现的一个重要标志。在 80C51 系列微控制器出现的年代，它的指令系统是很先进的。再强大的处理器也需要指令系统的配合才行。

3.1　概　　述

指令是微处理器控制计算机进行某种操作的命令，而指令系统则是全部指令的集合。计算机的功能是由其指令系统来实现的。一般来说，指令系统越丰富，计算机的功能也就越强。

3.1.1　指令分类

80C51 系列单片机共有 111 条指令，按其功能可分为 5 大类：数据传送类指令(28 条)、算术运算类指令(24 条)、逻辑运算类指令(25 条)、控制转移类指令(17 条)、布尔操作类指令(17 条)。

按指令代码的字节数可分为 3 大类：单字节指令(49 条)、双字节指令(45 条)、三字节指令(17 条)。

按指令的执行时间可分为 3 大类：单机器周期指令(64 条)、双机器周期指令(45 条)、四机器周期指令(2 条)。

3.1.2　指令格式

指令的表示方法称为指令格式，它包括指令的长度和指令内部信息的安排等。一条指令对应着一种基本操作，因此一条指令中通常包括操作性质和操作对象。例如“加”操作，操作对象是两个数，一个是被加数，另一个是加数，指令中除了要表达进行加法运算这一操作性质外，还必须指明参与操作的两个数或这两个数的存放地点(地址)以及相加结果应存放在何处。编写指令时采用的是助记符的形式，不能直接被硬件电路识别，需要通过编译程序将其转换为二进制代码，这种二进制代码称为指令代码或机器码，硬件电路根据机器码进行相应的操作。80C51 系列单片机的指令由操作码和操作数两大部分组成，格式可表示为：

[标号：] 操作码 [操作数 1][，操作数 2][，操作数 3][；注释]

[ ]表示其中内容是可选项。

(1) 标号是一条指令的标志，是可选项，与操作码之间用“：”隔开。

(2) 操作码指出了 CPU 应执行的操作类型，即操作性质。

(3) 操作数指出了参加操作的数据或数据的存放地址。它以一个或几个空格与操作码隔开。根据指令功能的不同，操作数可以有 1 个、2 个、3 个或者没有，操作数之间以“，”分隔。例如，指令“MOV A，B”属于双操作数指令；指令“POP A”属于单操作数指令；指令“RET”属于无操作数指令。

(4) 注释不属于指令执行部分，即不会被编译系统转换为机器码。一个可读性良好的程序需要有适当的注释，用来说明程序或指令的功能，便于程序的阅读和调试。用“；”作为指令与注释的分隔符。

3.1.3　指令中的符号

在说明和使用 80C51 系列 MCU 指令系统的功能时，经常使用一些符号，其意义如下。

A	累加器(ACC)。通常用 ACC 表示累加器的地址，用 A 表示它的名称。
B	寄存器。可以将立即数直接送给直接地址。
AB	累加器(ACC)和寄存器(B)组成的寄存器对。
direct	8 位片内 RAM 的存储单元地址。
#data	8 位立即数。



#datal6　　16 位立即数。

addr16　　16 位的地址码。

addr11　　11 位的地址码。

rel　　以补码表示的 8 位偏移量，其值为−128～+127。

bit　　片内 RAM 中可直接寻址的位地址。

Rn　　工作寄存器，其中 n=0～7。

Ri　　工作寄存器，其中 i=0～1。

@　　间接寻址符号。

+　　加。

−　　减。

*　　乘。

/　　除。

∧　　与。

∨　　或。

⊕　　异或。

=　　等于。

<　　小于。

>　　大于。

< >　　不等于。

←(→)　　表示数据传输方向。

(X)　　X(寄存器或 RAM 地址)单元的内容。

((X))　　以 X 单元的内容为地址的存储器单元内容。

$\overline{(X)}$　　X 寄存器的内容取反。

rrr　　指令代码中 rrr 三位的值由寄存器 Rn 确定，R7～R0 对应的 rrr 为 111～000。

$　　本条指令的起始地址。

3.2　寻 址 方 式

指令的一个重要组成部分是操作数，有些操作数不能直接给出，但可以给出操作数所存放的地址。指令给出寻找操作数的方式称为寻址方式。根据指令操作的需要，计算机有多种寻址方式。寻址方式越多，计算机功能就越强，灵活性就越大，但指令系统就越复杂。因此，寻址方式是否灵活方便是衡量指令系统好坏的重要指标。80C51 系列 MCU 指令系统的寻址方式共有 7 种。

3.2.1 立即寻址

这种寻址方式在指令中直接给出参与操作的常数(称为立即数)。立即数有 1 字节和 2 字节两种。

例如指令：

```
MOV          A, #5AH          ; A←5AH
```

其功能是将立即数 5AH 送入累加器 A 中。

又如指令：

```
MOV          DPTR, #2100H  ; DPTR←2100H
```

其功能是将 16 位立即数 2100H 送入 16 位寄存器 DPTR 中。

3.2.2 直接寻址

直接寻址是指在指令中给出操作数的直接地址，该地址指出了操作数所在的字节单元地址。

直接寻址方式可访问以下三种存储空间。

(1) 特殊功能寄存器(SFR)空间。SFR 只能用直接寻址方式访问。

(2) 片内 RAM 的低 128B 空间(00H～7FH)。

例如指令：

```
MOV          30H, 60H        ; 30H← (60H)
```

其功能是将片内 RAM 地址为 60H 的存储器单元的内容送到片内 RAM 地址为 30H 的存储器单元中。操作数 1 和操作数 2 都采用了直接寻址方式访问。

3.2.3 寄存器寻址

寄存器寻址是由指令给出某一寄存器的内容作为操作数。寄存器寻址对所选的工作寄存器区中 R7～R0 进行操作。指令代码中，所用寄存器的编码分别由 000～111 进行指示，累加器 A、寄存器对 AB 和 DPTR 也可用寄存器寻址方式访问，只是它们寻址时具体的寄存器编码隐含在指令代码中。

例如指令：

```
INC      R0        ; R0←(R0)+1，其功能是对 R0 进行操作，使其内容加 1
MOV      DPL, A    ; DPL←(A)，将 A 的内容传输到寄存器 DPL 中
MOV      A, P1     ; 此指令还可以写成“MOV   A, 90H”，90H 是 P1 口的直接地址
```

3.2.4 寄存器间接寻址

寄存器间接寻址是由指令指出某一寄存器的内容作为操作数的地址。需要注意的是，



此时寄存器的内容并不是操作数本身，而是存放操作数的(存储单元)地址。

80C51 系列 MCU 中可作为间接寻址的寄存器有：

(1) 片内 RAM 低 128B 单元，用选定的工作寄存器区中的 R0、R1 来寻址。

(2) 用堆栈指针 SP 来寻址的堆栈单元，即堆栈操作指令(PUSH 和 POP)的寻址方式。

(3) 用 R0、R1 或数据指针 DPTR 来寻址片外扩展的数据存储器。其中当存储单元的地址超过 8 位时需要用 DPTR 指示存储单元的地址。

例如指令：

```
MOV        A,@R0         ;A←((R0))
```

其功能是把指定的 R0 内容作为参与操作的数据的地址，把此地址单元的内容传至累加器 A 中，

寄存器间接寻址用符号“@”表示，以区别于寄存器寻址。

例如，将外部存储单元 2010H 中的内容传输到累加器 A 中，则指令为：

```
MOV        DPTR,#2010H  ;DPTR 中的内容为 2010H
MOVX       A, @DPTR     ;将 XRAM 的 2010H 单元的内容传输到 A 中
```

指令 MOVX 表明数据是从外部 RAM 传输到内部 RAM。传输过程如图 3-1 所示。

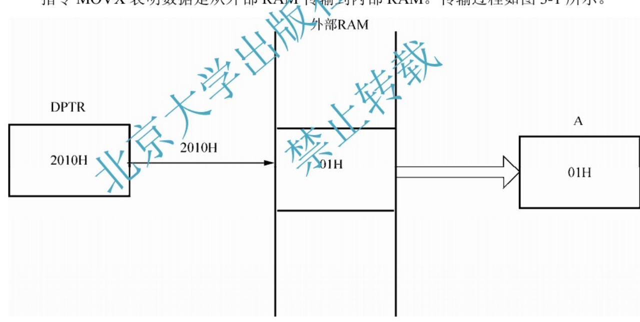


图 3-1　MOVX A, @DPTR 传输过程示例

3.2.5　变址寻址(基址寄存器+变址寄存器间接寻址)

变址寻址方式以 16 位的程序计数器 PC(当前地址)或数据指针 DPTR 作为基址寄存器，以 8 位的累加器 A 作为变址寄存器，基址寄存器内容和变址寄存器内容相加，其和形成 16 位的地址。该地址即为参与操作的数据的存储地址。

80C51 系列 MCU 共有 3 条变址寻址的指令：

```
MOVC        A, @A+PC        ; A←((A)+(PC))
MOVC        A, @A+DPTR      ; A←((A)+(DPTR))
JMP         @A+DPTR         ; PC←(A)+(DPTR)
```

前两条指令是从程序存储器区中取操作数，第 3 条指令是要获得程序的跳转地址，实现程序的转移。

例如指令：

```
MOV         A, #20H
MOV         DPTR, #2010H
MOVC        A, @A+DPTR
```

指令传输过程如图 3-2 所示。

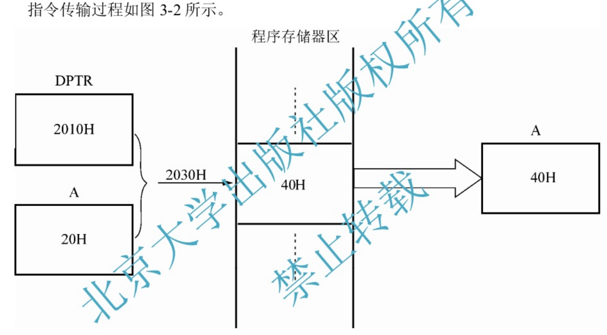


图 3-2　MOVC A, @A+DPTR 传输过程示例

3.2.6　相对寻址

相对寻址是把指令中规定的地址作为偏移量与 PC 的当前值相加，得到参与操作的数据的地址。这种寻址方式主要用于转移指令，指定转移的目标地址。

这种寻址方式与基址寄存器+变址寄存器的寻址方式不同，后者中的变址寄存器的内容是无符号数，而前者中的偏移量是有符号数，并以补码形式给出，所以转移的目标地址是在 PC 当前值的基础上加偏移量，也就是说，跳转的范围是在-128～+127 之间。

例如指令：

```
JC      #80H            ; 若(CY)=1, 则跳转
```

其功能是，当进位位(CY)=0 时，程序顺序执行；若(CY)=l，则以 PC 的当前值加上 80H 得到转移的目标地址。



若转移指令存放在 1005H 单元中(即 PC=1007H)，每取一字节的指令代码，程序计数器 PC 的内容自动加 1，而 JC 80H 的指令代码为两个字节，因此取出该条指令的代码(即执行完该条指令)后，PC 指向 1007H。这就是 PC 的当前值，把它与 80H(即－128)相加，就形成了目标地址，所以

目标地址=PC 当前值+偏移量= 1007H+0FF80H=0F87H

即程序将跳转到 0F87H 处执行。

3.2.7　位寻址

80C51 系列 MCU 有位处理功能，可以对数据位进行操作，因此就有相应的位寻址方式。位寻址的寻址范围如下。

1) 片内 RAM 中的位寻址区

片内 RAM 中的单元地址 20H～2FH，共 16 个单元 128 位，为位寻址区，位地址是 00H～7FH。对这 128 个位的寻址使用直接位地址表示。例如: MOV　C,2BH 指令的功能是把位寻址区的 2BH 位状态送入位累加器 C。

2) 可位寻址的特殊功能寄存器位

对于 80C51 系列 MCU 来说，字节地址可以被 8 整除的特殊功能寄存器中的每一位都可以按照位地址寻址，需要注意的是，可位寻址的是被定义的位，不可对未定义的位寻址。对这些寻址位在指令中有 4 种表示方法。我们以 PSW(字节地址 0D0H)中的位 5(位名称 F0)为例加以介绍。

(1) 直接使用位地址表示方法。例如：

```
MOV        C,OD5H          ;将位地址为 OD5H 的内容送入位累加器
```

② 存储单元地址加位的表示方法。例如：

```
MOV        C,ODOH.5        ;将字节地址为 ODOH 的位 5 的内容送入位累加器
```

③ 特殊功能寄存器符号加位的表示方法。例如：

```
MOV        C,PSW.5         ;将特殊功能寄存器 PSW 的位 5 的内容送入位累加器
```

④ 位名称表示方法，特殊功能寄存器中的一些寻址位是有名称的。例如：

```
MOV        C,FO            ;将用户标志位 FO 的内容送入位累加器
```

一个寻址位有多种表示方法，初看起来似乎复杂，实际上将为程序设计带来方便。一般来说，位名称方式是常用的方式。

对于指令中的操作数，因为指令操作常伴有从右向左传送数据的内容，所以常把左边操作数称为目的操作数，而右边操作数称为源操作数。上面所讲的各种寻址方式都是针对源操作数的，实际上，目的操作数也有寻址的问题。

例如：MOV 40H,R2 指令，其源操作数是寄存器寻址方式，而目的操作数则是直接寻址方式。上述指令的功能是，把按寄存器寻址方式取出的 R2 内容，再以直接寻址方式存放于内部 RAM 的 40H 单元中。

总的来说，源操作数的寻址方式多，而目的操作数的寻址方式较少，只有寄存器寻址、直接寻址、寄存器间接寻址和位寻址 4 种方式。因此，知道了源操作数的寻址方式，也就不难了解目的操作数的寻址问题了。

以上介绍了 80C51 系列 MCU 指令系统的 7 种寻址方式，概括起来见表 3-1。

表 3-1　7 种基本寻址方式及其相应的操作数寻址空间

序号	寻址方式	利用的变量	存储器空间
1	立即寻址	#data	数据存储器
2	直接寻址	direct	片内 RAM 低 128B 和 SFR
3	寄存器寻址	R0～R7,A,B,C,DPTR	工作寄存器和部分 SFR
4	寄存器间接寻址	@R0,@R1,@SP	片内 RAM
		@R0,@R1,@DPTR	片外 RAM 或 I/O 端口
5	变址寻址	@A+PC，@A+DPTR	程序存储器
6	相对寻址	PC+rel	程序存储器
7	位寻址	bit	片内 RAM 中位寻址区及可以位寻址的 SFR 位

3.3　指 令 系 统

3.3.1　数据传输类指令

数据传输操作是 CPU 最基本、最重要的操作之一。在程序中，数据传输指令占有相当大的比重。数据传输是否灵活、快速，对程序的编写和执行速度会产生很大影响。

1. 通用传输指令

指令格式：

```
MOV  <目的操作数>,<源操作数>
```

功能：把源操作数指定的内容传输到目的操作数指定的存储器单元中，指令不改变源操作数。

1) 以累加器 A 为目的地址的指令

```
指令                     操作
MOV      A,Rn           ;A←(Rn)
MOV      A,direct       ;A←(direct)
MOV      A,@Ri          ;A←((Ri))
MOV      A,#data        ;A←data
```

这组指令的功能是把源操作数指定的内容送入累加器 A 中。源操作数的寻址方式分别为寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。



例如：

```
MOV         A, R7          ; A←(R7)
MOV         A, 70H         ; A←(70H)
MOV         A, @R0         ; A←((R0))
MOV         A, #80H        ; A←80H
```

2) 以 Rn 为目的地址的指令

指令　　　　　　　　　　操作

```
MOV     Rn, A              ; Rn←(A)
MOV     Rn, direct         ; Rn←(direct)
MOV     Rn, #data          ; Rn←data
```

这组指令的功能是把源操作数指定的内容送入工作寄存器区 R7～R0 中的某一寄存器中。源操作数的寻址方式分别为寄存器寻址、直接寻址和立即寻址。

例如：

```
MOV     R3, A              ; R3←(A)
MOV     R7, 70H            ; R7←(70H)
MOV     R5, #0FAH          ; R5←0FAH
```

3) 以直接地址为目的地址的指令

指令　　　　　　　　　　操作

```
MOV     direct, A          ; direct←(A)
MOV     direct, Rn         ; direct←(Rn)
MOV     direct, @Ri        ; direct←((Ri))
MOV     direct1, direct2   ; direct1←(direct2)
MOV     direct, #data      ; direct←data
```

这组指令的功能是将源操作数指定的内容送入由直接地址给出的内部 RAM 存储单元中。源操作数的寻址方式分别为寄存器寻址、寄存器间接寻址、直接寻址和立即寻址等。

例如：

```
MOV     P1, A              ; P1←(A), 等价于 MOV   90H, A
MOV     70H, R3            ; 70H←(R3)
MOV     30H, @R0           ; 30H←((R0))
MOV     0E0H, 78H          ; 0E0H←(78H)
MOV     01H, #50H          ; 01H←50H, 等价于 MOV   R1, #50H
```

4) 以寄存器间接地址为目的地址的指令

指令　　　　　　　　　　操作

```
MOV     @Ri, A             ; (Ri)←(A)
MOV     @Ri, direct        ; (Ri)←(direct)
MOV     @Ri, #data         ; (Ri)←data
```

这组指令的功能是把源操作数指定的内容送入R0或R1指出的片内RAM存储单元中。源操作数的寻址方式分别为寄存器寻址、直接寻址和立即寻址。

例如：

```
MOV    @R1, A          ; R1←(A)
MOV    @R0, 70H        ; R0←(70H)
MOV    @R1, #78H       ; R1←78H
```

5) 16位数据传输指令

```
指令                    操作
MOV    DPTR, #data16    ; DPTR←data16
```

这条指令的功能是把一个16位的立即数送入DPTR中。16位的数据指针DPTR由DPH和DPL组成。这条指令把立即数的高8位送DPH，低8位送DPL。

上述指令中，累加器A是一个特别重要的8位寄存器，Rn为CPU当前选择的工作寄存器区中的R7～R0，直接地址指出的存储单元为片内RAM的00H～7FH和特殊功能寄存器SFR，在间接寻址中，用R0和R1作为地址指针访问片内RAM的00H～7FH(89C52为00H～FFH)这128(89C52为256)个单元。

例3.1 设(70H)=60H，(60H)=20H，P1口为输入口，当前的输入状态为0B7H，执行下面程序：

```
MOV    R0, #70H      ; R0←70H
MOV    A, @R0        ; A←60H
MOV    R1, A         ; R1←60H
MOV    A, @R1        ; A←20H
MOV    @R0, P1       ; (70H)←0B7H
```

结果为：(70H)=0B7H, (A)=20H, (R1)=60H, (R0)=70H。

2. 片外数据存储器(或扩展I/O口)的传输指令

指令格式：

```
MOVX  <目的操作数>, <源操作数>
```

功能：实现片外数据存储器(或扩展I/O口)与累加器A之间的数据传输。寻址方式只能用间接寻址。这组指令有：

```
指令                    操作
MOVX   A, @DPTR       ; A←((DPTR))
MOVX   A, @Ri         ; A←((Ri))
MOVX   @DPTR, A       ; (DPTR)←(A)
MOVX   @Ri, A         ; (Ri)←(A)
```

由于片外扩展的RAM和I/O口是统一编址的，共同使用64 KB的存储空间，所以由



指令本身看不出是对片外 RAM 还是对扩展 I/O 口进行操作，而是由硬件的地址分配来定。

用 Ri 进行间接寻址时，因为 Ri 是一个 8 位寄存器，用它只能寻址 256 个存储单元。当片外 RAM 容量小于 256 B 时，可直接采用这种寻址方式；当片外 RAM 超过 256 B 时，就要利用 P2 口输出高 8 位地址(也称页地址)，而由@Ri 进行页内(每 256 个单元为 1 页)寻址。

例如，若要把片外 RAM 的 2010H 单元的内容传输到累加器 A 中，则可采用以下指令：

```
MOV     P2, #20H          ; P2←20H, 得到页地址
MOV     R0, #10H          ; R0←10H, 得到页内地址
MOVX    A, @R0            ; A←(2010H)
```

当然，也可使用 DPTR 进行间接寻址，完成上述功能的指令为：

```
MOV     DPTR, #2010H      ; DPTR←2010H, 得到片外 RAM 的地址
MOVX    A, @DPTR          ; A←(2010H)
```

3. 程序存储器向累加器 A 传输指令

指令格式：

```
MOVC  A, <源操作数>
```

功能：从程序存储器中读取源操作数指定的内容送入累加器 A 中。寻址方式只能采用变址寻址。这组指令包括以下两条指令。

```
      指令                       操作
MOVC    A, @A+PC          ; A←((A)+(PC))
MOVC    A, @A+DPTR        ; A←((A)+(DPTR))
```

第一条指令以 PC 作为基址寄存器，A 的内容(无符号数)作为变址与 PC 当前值相加得到一个 16 位的地址，将该地址给出的程序存储器单元的内容送入累加器 A 中。

例如，设(A)=50H，PC 的当前值为 1001H，执行指令；

```
MOVC    A, @A+PC
```

则执行结果是 PC 的当前值 1001H 与 A 中内容 50H 相加得 1051H，然后将程序存储器中该值对应单元的内容送入累加器 A 中。

第二条指令以 DPTR 为基址寄存器，以 A 的内容(无符号数)作为变址与 DPTR 的内容相加得到一个 16 位地址，将该地址指出的程序存储器单元的内容送入累加器 A 中。

例 3.2　编写根据累加器 A 中的数(0～9)查平方表的程序。

解：把平方表用伪指令 DB 存放在程序存储器中，把表的首地址置入 DPTR 中，把数 0～9 存放在累加器 A 中，程序如下：

```
MOV             DPTR, #TABLE      ; TABLE 表示表首地址
MOVC            A, @A+DPTR
TABLE：DB   00H, 01H, 04H, 09H, 10H, 19H, 24H, 31H, 40H, 51H
```

这组指令常用于程序存储器中的查表操作，故也称作查表指令，是 80C51 系列 MCU 的特色指令之一。其中，“MOVC A,@A+PC”指令称为近程查表指令(因为它只能在以 PC 当前值为基准的+256 B 范围内查表)，而“MOVC A, @A+DPTR”称为远程查表指令(它可以在 64 KB 范围内查表)。

4. 数据交换指令

1) 字节交换指令 XCH

指令　　　　　　　　　　操作

```
XCH    A,Rn        ;(A)←→(Rn)
XCH    A,direct    ;(A)←→(direct)
XCH    A,@Ri       ;(A)←→((Ri))
```

这组指令的功能是将累加器 A 的内容和源操作数指定的内容相互交换。源操作数的寻址方式分别为寄存器寻址、直接寻址和寄存器间接寻址。

例 3.3　设(A)=80H，(R7)=08H，执行指令：

```
XCH    A,R7
```

结果为：(A)=08H，(R7)=80H。

2) 半字节交换指令 XCHD

指令　　　　　　　　　　操作

```
XCHD   A,@Ri       ;(A)3~0←→((Ri))3~0
```

这条指令的功能是将 A 的低 4 位和(R0)、(R1)给出的片内 RAM 单元的低 4 位相互交换，各自的高 4 位不变。

例 3.4　设(A)=15H，(R0)=30H，(30H)=34H，执行指令：

```
XCHD   A,@R0
```

结果为：(A)=14H，(30H)=35H，(R0)=30H。

5. 堆栈操作指令

在 80C51 系列 MCU 的片内 RAM 中，可以设置一个后进先出(LIFO)的堆栈，特殊功能寄存器 SP 作为堆栈指针，在进行堆栈操作时，始终指向栈顶所在的位置。在指令系统中有两组用于数据传输的栈操作指令。

1) 进栈(压栈)指令 PUSH

指令　　　　　　　　　　操作

```
PUSH   direct      ;SP←(SP)+1
                   ;(SP)←(direct)
```

这组指令的功能是首先将堆栈指针 SP 的内容加 1，然后把直接地址给出的内容传输到堆栈指针(SP)所寻址的片内 RAM 单元中。



例 3.5　设(SP)=60H，(A)=10H，(B)=50H，若执行指令：

```
PUSH    ACC         ; SP←(SP)+1=61H, (SP)←(ACC)
PUSH    B           ; SP←(SP)+1=62H, (SP)←(B)
```

结果(61H)=10H，(62H)=50H，(SP)=62H，执行情况如图 3-3 所示。

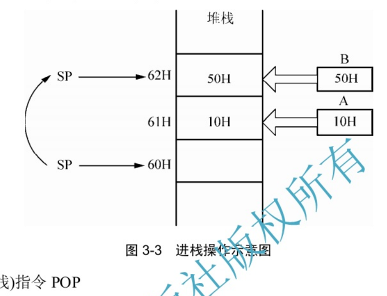


图 3-3　进栈操作示意图

2) 出栈(弹栈)指令 POP

指令　　　　　　　　操作

```
POP     direct      ;direct←((SP))
                    ;SP←(SP)-1
```

这组指令的功能是将堆栈指针(SP)寻址的片内 RAM 单元内容送入直接地址指出的存储单元中，然后 SP 的内容减 1。

例 3.6　设(SP)=62H, (62H)=50H, (61H)=10H，若执行指令：

```
POP     DPH         ; DPH←((SP))=(62H), SP←(SP)-1=61H
POP     DPL         ; DPL←((SP))=(61H), SP←(SP)-1=60H
```

结果为：(DPTR)=5010H，(SP)=60H，执行情况如图 3-4 所示。

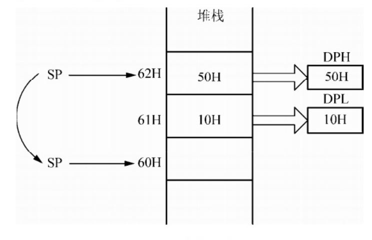


图 3-4　出栈示意图

80C51 系列 MCU 的数据传输指令中，除了“POP”指令和直接将数据送到程序状态字 PSW 的“MOV”指令及以累加器 A 为目的地址的传输指令影响 P 标志外，其余指令均不影响标志。

3.3.2 算术运算类指令

80C51 系列单片机的算术运算指令有加、减、乘、除法指令和加 1、减 1 指令。

1. 加法指令

1) 不带进位的加法指令 ADD

```
指令                          操作
ADD     A,Rn              ;A←(A)+(Rn)
ADD     A,direct          ;A←(A)+(direct)
ADD     A,@Ri             ;A←(A)+((Ri))
ADD     A,#data           ;A←(A)+data
```

这组加法指令的功能是将源操作数指定的内容和累加器 A 的内容相加，结果存放在 A 中，若 D7 位产生进位，则进位位 CY 被置 1，否则 CY 被清 0。若 D3 位产生进位，则辅助进位位 AC 被置 1，否则 AC 被清 0。对溢出标志 OV 的影响是：如果 D6 位有进位而 D7 位无进位，或者 D7 位有进位而 D6 位无进位，则 OV 标志被置 1，否则被清 0。从另一方面看，若把参加运算的数看作 8 位二进制补码，当运算结果超过二进制补码所表示的范围(+127～－128)时，OV 被置 1，否则 OV 被清 0。在进行符号数的补码运算时，溢出标志 OV=1，表示运算结果出错。

奇偶标志位 P 将随累加器 A 中 1 的个数的奇偶性的变化而变化。80C51 系列 MCU 的 CPU 的校验方式为偶校验，即，若 A 中 1 的个数为奇数，则 P 置 1，否则 P 为 0。

源操作数的寻址方式可采用寄存器寻址、直接寻址、寄存器间接寻址和立即寻址方式。

例 3.7 设(A)=89H, (50H)=9AH，执行指令：

```
ADD         A,50H
```

结果为：(A)=23H，(CY)=1，(AC)=1，(OV)=1，(P)=1。

例 3.8 设(A)=52H，(R0)=20H，(20H)=0FCH，执行指令：

```
ADD         A,@R0
```

结果为：(A)=4EH，(CY)=1，(AC)=0，(OV)=0，(P)=0。

2) 带进位加法指令 ADDC

```
指令                      操作
ADDC    A,Rn              ;A←(A)+(Rn)+(CY)
ADDC    A,direct          ;A←(A)+(direct)+(CY)
ADDC    A,@Ri             ;A←(A)+((Ri))+(CY)
ADDC    A,#data           ;A←(A)+data+(CY)
```



这组指令的功能是将源操作数指定的内容、进位位 CY 和累加器 A 的内容相加，结果存放在 A 中。

ADDC 指令对 PSW 标志位的影响与 ADD 指令的影响相同。这组指令多用于多字节加法运算。在进行高字节加法运算时，要考虑低位字节向高位字节进位的情况。

例 3.9　设(A)=48H，(R3)=62H，(CY)=1，执行指令：

```
ADDC    A, R3
```

结果为：(A)=0ABH，(CY)=0，(AC)=0，(OV)=l，(P)=1。

3) 加 1 指令 INC

```
指令                    操作
INC     A              ; A←(A)+l
INC     Rn             ; Rn←(Rn)+l
INC     direct         ; direct←(direct)+l
INC     @Ri            ; (Ri)←((Ri))+1
INC     DPTR           ; DPTR←(DPTR)+1
```

这组指令的功能是把操作数所指出的内容加 1。若原来为 0FFH，将产生溢出(上溢)，结果为 00H。本组指令除“INC　A”指令影响 P 标志外，其余均不影响任何标志位。操作数的寻址方式可采用寄存器寻址、直接寻址或寄存器间接寻址方式。当用本组指令修改输出端口数据时，原端口数据的值将从端口锁存器 P0～P3 读入，而不是从引脚读入，加 1 后再存入端口锁存器。

例 3.10　设(A)=0FFH，(R5)=0FH，(38H)=0F0H，(R0)=39H，(39H)=00H，执行指令：

```
INC    A
INC    R5
INC    38H
INC    @R0
```

结果为：(A)=00H，(R5)=10H，(38H)=0F1H，(39H)=01H，(P)=0。

4) 十进制调整指令 DA

```
DA          A
```

这条指令可对压缩 BCD 码(一个字节存放 2 位 BCD 码)的加法结果进行十进制调整。两个压缩 BCD 码按二进制相加后，必须经过十进制调整才能得到正确的压缩 BCD 码的和数。

例 3.11　设(A)=56H，(R6)=67H，执行指令：

```
ADD         A, R6
DA          A
```

结果为：(A)=23H，(CY)=1。

2. 减法指令

1) 带借位减法指令 SUBB

```
指令                        操作

SUBB    A, Rn              ; A←(A) - (Rn) - (CY)
SUBB    A, direct          ; A←(A) - (direct) - (CY)
SUBB    A, @Ri             ; A←(A) - ((Ri)) - (CY)
SUBB    A, #data           ; A←(A) - data- (CY)
```

这组指令的功能是从累加器 A 中减去源操作数指定的内容及进位标志位 CY 的内容，结果存放在 A 中。

运算结果中，若 D7 位有借位，则置(CY)=1，否则(CY)=0。若 D3 位有借位，则(AC)=1，否则(AC)=0。若 D6 位有借位而 D7 位没有借位，或 D7 位有借位而 D6 位没有借位，则溢出标志(OV)=1，否则(OV)=0。

源操作数的寻址方式可采用寄存器寻址、直接寻址、寄存器间接寻址和立即寻址等方式。

例 3.12 设(A)=0F8H，(R0)=66H，(CY)=1。执行指令：

```
SUBB    A, R0
```

结果为：(A)=91H，(CY)=0，(AC)=0，(OV)=0，(P)=1。

请注意，由于 80C51 系列 MCU 没有不带借位的减法指令，若需进行不带借位的减法运算，则应该先将 CY 清 0，然后再执行 SUBB 指令。

2) 减 1 指令 DEC

```
指令                   操作

DEC     A             ; A←(A) - 1
DEC     Rn            ; Rn←(Rn) - 1
DEC     direct        ; direct←(direct) - 1
DEC     @Ri           ; Ri←((Ri)) - 1
```

这组指令的功能是将操作数指定的内容减 1。若原来为 00H，减 1 后将产生溢出(下溢)，结果为 0FFH。除“DEC　A”指令影响 P 标志外，其余均不影响任何标志位。操作数的寻址方式可采用寄存器寻址、直接寻址或寄存器间接寻址方式。当用本组指令修改输出端口数据时，原端口数据的值将从端口锁存器 P0～P3 读入，而不是从引脚读入，减 1 后再存入端口锁存器。

例 3.13 设(A)=2FH，(R7)=39H，(20H)=00H，(R1)=21H，(21H)=0FFH，执行指令：

```
DEC     A
DEC     R7
DEC     20H
DEC     @R1
```



结果为：(A)=2EH，(R7)=38H，(20H)=0FFH，(21H)=0FEH，(P)=0。

3. 乘法指令 MUL

```
MUL          AB
```

这条指令是把累加器 A 和寄存器 B 中的无符号 8 位二进制数相乘，乘积的低 8 位存放在累加器 A 中，高 8 位存放在寄存器 B 中。

如果乘积大于 0FFH，则(OV)=1，否则(OV)=0。CY 标志总是被清 0。

例 3.14 设(A)=60H，(B)=0B0H，执行指令：

```
MUL          AB
```

结果为：(B)=42H，(A)=00H(即积为 4200H)，(OV)=1。

4. 除法指令 DIV

```
DIV          AB
```

这条指令的功能是把累加器 A 中的 8 位无符号二进制数(被除数)除以寄存器 B 中的 8 位无符号二进制数(除数)，所得的商(整数部分)存放在累加器 A 中，余数部分存放在寄存 B 中。

如果寄存器 B 中原来的内容为 0，即除数为 0，则结果 A 和 B 的内容不定，且溢出标志位(OV)=1。CY 标志总是被清 0。

例 3.15 (A)=0FAH，(B)=24H，执行指令：

```
DIV          AB
```

结果为：(A)=06H，(B)=22H，(CY)=0，(OV)=0。

3.3.3 逻辑运算类指令

1. 单操作数的逻辑运算指令

1) 清 0 指令

指令		操作
CLR	A	; A←00H

这条指令的功能是将累加器 A 清 0，只影响 P 标志位。

2) 取反指令

指令		操作
CPL	A	; A←$\overline{(A)}$

这条指令的功能是将累加器 A 的每一位逻辑取反，只影响 P 标志位。

例 3.16 设(A)=01010101B，执行指令：

```
CPL          A
```

结果为：(A)=10101010B。

3) 循环左移指令

```
RL        A
```

这条指令的功能是将累加器 A 的内容向左循环移 1 位，将 ACC.7 移入 ACC.0 中，不影响标志位。

4) 带进位循环左移指令

```
RLC       A
```

这条指令的功能是将累加器 A 的内容和进位标志 CY 的内容一起向左循环移 1 位，将 ACC.7 移入 CY 中，CY 移入 ACC.0 中，不影响其他标志位。

5) 循环右移指令

```
RR        A
```

这条指令的功能是将累加器 A 的内容向右循环移 1 位，ACC.0 移入 ACC.7 中，不影响标志位。

6) 带进位循环右移指令

```
RRC       A
```

这条指令的功能是将累加器 A 的内容和进位标志 CY 的内容一起向右移 1 位，ACC.0 移入 CY 中，CY 移入 ACC.7 中，不影响其他标志位。

7) 累加器半字节交换指令

```
SWAP  A
```

这条指令的功能是将累加器 A 的高 4 位与低 4 位互换，不影响标志位。

例如，若(A) =00010010B，(CY) =1，则执行下列指令：

```
RL        A         ;(A) =00100100,不影响标志位 CY
RLC       A         ;(A) =00100101,(CY)=0
RR        A         ;(A) =00001001,不影响标志位 CY
RRC       A         ;(A) =10001001,(CY)=0
```

2. 双操作数的逻辑操作指令

1) 逻辑与运算指令

```
指令                  操作
ANL    A,Rn           ;A←(A)∧(Rn)
ANL    A,direct       ;A←(A)∧(direct)
ANL    A,@Ri          ;A←(A)∧((Ri))
ANL    A,#data        ;A←(A)∧data
ANL    direct,A       ;direct←(direct)∧(A)
ANL    direct,#data   ;direct←(direct)∧data
```



这组指令的功能是将源操作数和目的操作数所指定的内容按位进行逻辑与运算，结果存放在目的操作数所指定的地址中。源操作数的寻址方式可采用寄存器寻址、直接寻址、寄存器间接寻址和立即寻址方式。当用本组指令修改输出端口数据时，原端口数据的值将从端口锁存器 P0～P3 读入，而不是从引脚读入，运算结果存入端口锁存器。除前 4 条指令会影响 P 标志位外，这组指令不影响其他标志位。

例 3.17　设(A)=05H，(R7)=73H，执行指令：

```
ANL        A,R7
```

结果为：(A)=01H，(P)=1。

2) 逻辑或运算指令

```
指令                          操作
ORL     A,Rn                ;A←(A)∨(Rn)
ORL     A,direct            ;A←(A)∨(direct)
ORL     A,@Ri               ;A←(A)∨((Ri))
ORL     A,#data             ;A←(A)∨data
ORL     direct,A            ;direct←(direct)∨(A)
ORL     direct,#data        ;direct←(direct)∨data
```

这组指令的功能是将源操作数和目的操作数所指定的内容按位进行逻辑或运算，结果存放在目的操作数所指定的地址中。源操作数的寻址方式同样可采用寄存器寻址、直接寻址、寄存器间接寻址和立即寻址方式。同 ANL 指令类似，用于修改输出端口数据时，原端口数据的值将从端口锁存器 P0～P3 读入。前 4 条指令只影响 P 标志位。

例 3.18　设(P3)=05H，(A)=73H，执行指令：

```
ORL        P3,A
```

结果为：(P3)=77H。

3) 逻辑异或指令

```
指令                          操作
XRL     A,Rn                ;A←(A)⊕(Rn)
XRL     A,direct            ;A←(A)⊕(direct)
XRL     A,@Ri               ;A←(A)⊕((Ri))
XRL     A,#data             ;A←(A)⊕data
XRL     direct,A            ;direct←(direct)⊕(A)
XRL     direct,#data        ;direct←(direct)⊕data
```

这组指令的功能是将源操作数和目的操作数所指定的内容按位进行逻辑异或运算，结果存放在目的操作数所指定的地址中。源操作数的寻址方式同样可采用寄存器寻址、直接寻址、寄存器间接寻址和立即寻址方式。与 ANL 指令类似，用于修改输出端口数据时，原端口数据的值将从端口锁存器 P0～P3 读入。前 4 条指令只影响 P 标志位。

例 3.19 设(A)=05H，(R5)=73H，执行指令：

```
XRL            A,R5
```

结果为：(A)=76H，(P)=1。

3.3.4 控制转移类指令

1. 无条件跳转指令

1) 短跳转指令

指令　　　　　　　　　　　　　操作

```
AJMP   addr11          ;PC←(PC)+2, PC10~0←addr11
```

在运行该指令时，将 PC 当前值的高 5 位和 addr11 的 11 位地址相连($PC_{15\sim11}$ $addr_{10\sim0}$)，得到的 16 位转移目标地址送入 PC。由于指令只提供了 11 位地址，而高 5 位地址不变，因此转移范围在 2 KB 地址空间内。也就是说，目标地址必须写在它下一条指令存放地址的同一个 2 KB 区域内。

2) 相对转移指令

指令　　　　　　　　　　　　　操作

```
SJMP   rel             ;PC←(PC)+2, PC←(PC) +rel
```

这也是一种无条件转移指令，转移的目标地址由 PC 当前值加上相对偏移量 rel 组成。rel 是以补码形式表示的 8 位有符号二进制数，因此本指令的转移范围为：PC 当前值-128～PC 当前值+127。

3) 长跳转指令

指令　　　　　　　　　　　　　操作

```
LJMP   addr16          ;PC←addr16
```

这条指令的功能是将指令提供的16位地址送入PC中，然后程序无条件转向目标地址。由于指令提供了 16 位地址，因此程序可以转向 64 KB 的程序存储器地址空间内的任何单元。

4) 基址寄存器加变址寄存器间接转移指令(散转指令)

指令　　　　　　　　　　　　　操作

```
JMP    @A+DPTR         ;PC←(PC)+1, PC←(A)+(DPTR)
```

这条指令的功能是将累加器 A 中 8 位无符号二进制数与数据指针 DPTR 的内容相加，结果送入 PC 作为下次执行指令的地址。该指令不影响标志位。利用这条指令可实现程序的散转，可以代替许多判别跳转指令。

2. 条件转移指令

条件转移就是程序的转移是有条件的，当条件满足时，进行转移，否则顺序执行下一条指令。转移的目标地址在以 PC 当前值为中心的 256B 范围内(PC-128～PC+127)。



1) 测试条件符合转移指令

指令		操作
JZ	rel	; 若(A) =0, 则 PC←(PC) +2+ rel
JNZ	rel	; 若(A) ≠0, 则 PC←(PC) +2+ rel

2) 比较不相等转移指令

指令		操作
CJNE	A, direct, rel	; 若(A) ≠(direct), 则 PC←(PC) +3+ rel
CJNE	A, #data, rel	; 若(A) ≠data, 则 PC←(PC) + 3+ rel
CJNE	Rn, #data, rel	; 若(Rn) ≠data, 则 PC←(PC) + 3+ rel
CJNE	@Ri, #data, rel	; 若((Ri)) ≠data, 则 PC←(PC) + 3+ rel

这组指令的功能是比较指令中两个操作数的值是否相等。如果两数不相等，则转移，转移的目标地址为 PC 当前值与偏移量 rel 相加所得地址，否则顺序执行下一条指令。如果两数比较不相等，且操作数 1(无符号数)小于操作数 2，则(CY)=l，否则(CY)=0。该组指令不影响任何操作数的内容及其他标志位。

3) 减 l 不为 0 转移指令

指令		操作
DJNZ	Rn, rel	; Rn←(Rn) - 1, 若(Rn) ≠0, 则 PC←(PC) +2+ rel
DJNZ	direct, rel	; direct←(direct) - 1, 若(direct) ≠0, 则 PC←(PC) +3+ rel

这组指令的功能是将操作数指定的内容减 l，结果仍送回操作数所指定的地址中。如果结果不为 0，则转移，目标地址为 PC 当前值与偏移量 rel 相加所得地址，否则顺序执行下一条指令。

这组指令常用于循环计数，允许使用者把片内 RAM 单元用作程序循环计数器。

例如，执行如下程序：

```
        MOV     R0, #0AH
        CLR     A
AGAIN:  ADD     A, R0
        DJNZ    R0, AGAIN     ; 将 R0 的内容减 1, 若不为 0 则跳转到 AGAIN
        SJMP    $
```

结果为：(A)=10+9+8+…+1=37H。

指令“SJMP　$”表示退出循环后“原地等待”。“$”表示本条指令所在地址，这条指令也可以写成：“WAIT：SJMP　WAIT”，功能相同。

3. 调用和返回指令

在程序设计中，常常会出现几个地方都需要进行功能完全相同的处理的情况。为了减少程序编写和调试的工作量，使某一段程序能被公用，引入了主程序和子程序的概念。

通常把具有一定功能的公用程序段作为子程序单独编写，当主程序需要调用这个子程

序时，可利用调用指令对子程序进行调用。在子程序末尾安排一条返回指令，使子程序执行结束后能返回到主程序。

在一个程序中，往往子程序还会调用其他子程序，这称为子程序嵌套。每次调用子程序时，必须将 PC 当前值(断点地址)压入堆栈保存起来，以便返回时按“后进先出”的原则依次取出原来保存的 PC 值。调用指令和返回指令分别具有自动保护和恢复 PC 内容的功能。

1) 短调用指令

```
ACALL     addr11
```

执行该指令时，堆栈指针 SP 的值加 2，将 PC 当前值压入堆栈(先 PC 的低 8 位，后 PC 的高 8 位)，然后把 PC 的高 5 位与 addr11 的 11 位地址相连接($PC_{15\sim11}$ $addr_{10\sim0}$)，获得子程序的 16 位地址并送入 PC 中，使 CPU 转向执行子程序。

该指令的操作过程为：SP←(SP)+1，(SP)←($PC_{7\sim0}$)，SP←(SP)+1，(SP)←($PC_{15\sim8}$)，PC←$PC_{15\sim11}$ $addr_{10\sim0}$。

该指令所调用的子程序的入口地址必须在 ACALL 指令后的 2KB 区域内。

例 3.20 设(SP)=60H，标号 MA 值为 0123H，子程序位于 0345H，执行指令：

```
MA: ACALL     0345H
```

结果为：(SP)=62H，(61H)=25H，(62H)=01H，(PC)=0345H。

2) 长调用指令

```
LCALL     addr16
```

执行该指令时，堆栈指针 SP 的值加 2，将 PC 当前值压入堆栈，然后把 addr16 的 16 位地址送入 PC 中，使 CPU 转向执行子程序。

该指令的操作过程为：SP←(SP)+1，(SP)←($PC_{7\sim0}$)，SP←(SP)+1，(SP)←($PC_{15\sim8}$)，PC←addr16。

该指令可调用 64 KB 范围内程序存储器中任何一个子程序，执行后不影响任何标志位。

例 3.21 设(SP)=60H，标号 STAR 值为 0100H，标号 DIR 值为 6789H，执行指令：

```
STAR: LCALL     DIR    (或  STAR: LCALL     6789H)
```

结果为：(SP)=62H，(61H)=03H，(62H)=01H，(PC)=6789H。

3) 返回指令

如上所述，返回指令是使 CPU 从子程序或中断服务程序返回执行主程序的指令。

(1) 从子程序返回指令。

```
RET
```

操作：PCH←((SP))，SP←(SP)−1，PCL←((SP))，SP←(SP)−1。使 CPU 从堆栈中弹出的 PC 值处开始执行程序。该指令不影响任何标志位。



例 3.22　设(SP)=62H，(62H)=07H，(61H)=30H，执行指令：

```
RET
```

结果为：(SP)=60H，(PC)=0730H。

在子程序的末尾必须有一条返回指令，才能使 CPU 从子程序中返回主程序执行。

(2) 中断返回指令。

```
RETI
```

该指令除了执行 RET 指令的操作外，还将 MCU 内部的中断优先级寄存器的相应中断优先级标志清 0(此状态位在中断响应时被置 1)。因此，中断服务程序必须以 RETI 为结束指令。有关中断处理过程将在后续章节中进行详细介绍。

请读者注意，在子程序或中断服务子程序中，PUSH 指令必须与 POP 指令成对使用，否则，不能正确返回主程序。

4. 空操作指令

```
NOP
```

执行该指令使 PC 值加 1，然后继续执行下条指令。本指令无任何操作，它为单周期指令，在时间上消耗一个机器周期，因而常用于延时或等待程序中“微调”时间。

3.3.5　位操作类指令

80C51 系列 MCU 中设置了独立的布尔处理器。布尔处理器有自己相应的位累加器 CY、存储器和 I/O 口等。布尔处理器也有自己丰富的位操作指令，包括位数据传输、位状态控制、位逻辑操作和位控制转移操作指令等。

1. 位数据传输指令

```
指令                 操作
MOV    C,bit        ;CY←(bit)
MOV    bit,C        ;bit←(CY)
```

这组指令的功能是在以 bit 表示的位地址和进位位 CY 之间进行数据传输，不影响其他标志位。

例如，(06H)=1，下列指令及操作结果为：

```
MOV    C,06H        ;CY←(20H.6)
MOV    P1.0,C       ;P1.0←(CY)
```

结果为：(P1.0)=1。

2. 位状态控制指令

位状态控制指令共有 6 条。

指令		操作
CLR	C	; CY←0
CLR	bit	; bit←0
CPL	C	; CY←$\overline{(CY)}$
CPL	bit	; bit←$\overline{(bit)}$
SETB	C	; CY←1
SETB	bit	; bit←1

这组指令的功能是将操作数指出的位清零，取反、置 1，不影响其他标志位。

例如，下列指令及操作结果为：

CLR	C	; CY←0
CLR	ACC. 7	; ACC. 7←0, 将累加器的位 7 清 0
CPL	2AH. 0	; 2AH. 0←$\overline{(2AH. 0)}$, 将 2AH 字节单元的第 0 位取反
SETB	P2. 7	; P2. 7←1, 将 P2 口寄存器的位 7 置 1

3. 位逻辑操作指令

1) 位逻辑与操作指令

指令		操作
ANL	C, bit	; CY←(CY) ∧ (bit)
ANL	C, /bit	; CY←(CY) ∧ $\overline{(bit)}$

这组指令是将指定的位地址单元内容(或取反后的内容)与位累加器 CY 内容进行逻辑与操作，结果送入 CY 中，指定的位地址单元内容不变，不影响其他标志位。

2) 位逻辑或操作指令

指令		操作
ORL	C, bit	; CY←(CY) ∨ (bit)
ORL	C, /bit	; CY←(CY) ∨ $\overline{(bit)}$

这组指令与 ANL 指令类似，是将指定位地址单元中的内容(或取反后的内容)与位累加器 CY 进行逻辑或操作，结果送入 CY 中，不影响其他标志位。

4. 位控制转移操作指令

位状态控制指令共有 6 条。

1) 位累加器条件转移指令

指令		操作
JC	rel	; 若(CY) =1, 则 PC←(PC) +2+ rel
JNC	rel	; 若(CY) =0, 则 PC←(PC) +2+ rel

指令的功能是对位累加器 CY 进行检测，当(CY)=1 或(CY)=0 时，程序转向 PC 当前值与偏移量 rel 之和的目标地址，否则顺序执行下一条指令。因此转移的范围是 PC−128～PC+127。操作不影响标志位。



例如：设(CY)= 0。执行指令

```
JC          LABEL1
CPL         C
JC          LABEL2
```

则执行结果为：进位位取反变为 1，程序转向 LABEL2 地址单元执行。

2) 位状态条件转移指令

```
    指令                    操作
    JB      bit,rel         ;若(bit)=1，则 PC←(PC)+3+ rel
    JNB     bit,rel         ;若(bit)=0，则 PC←(PC)+3+ rel
```

指令的功能是检测指定位 bit 的内容，当位状态分别为 1 或 0 时，程序转向 PC 当前值与偏移量 rel 之和的目标地址，否则顺序执行下一条指令。因此转移的范围是 PC-128～PC+127。操作不影响标志位。

例如：设累加器 A 中的内容为 8EH(10001110B)。执行指令

```
JB          ACC.0,LABEL1
JB          ACC.1,LABEL2
```

则执行结果为：程序转向 LABEL2 单元执行。

3) 位状态条件转移并清零指令

```
    指令                    操作
    JBC     bit，rel        ;若(bit)=1,则 PC←(PC)+3+ rel，bit←0
```

指令的功能是检测指定位 bit 的内容，当位状态为 1 时，则将该位清零，并且程序转向 PC 当前值与偏移量 rel 之和的目标地址，否则顺序执行下一条指令。因此转移的范围是 PC-128～PC+127。操作不影响标志位。

例如：设累加器 A 中的内容为 7AH(01111010B)。执行指令

```
JBC         ACC.7,LABEL1
JBC         ACC.6,LABEL2
```

则执行结果为：程序转向 LABEL2 单元执行，并将 ACC.6 位清零，(A)=3AH(00111010B)。

3.4　汇 编 语 言

3.4.1　程序设计语言概述

在计算机的程序设计中，可以使用三种语言来编写程序。

1. 机器语言

机器语言是用二进制数(十六进制数)表示的指令和数据的总称，或称机器代码、指令

代码。用机器语言编写的程序称为目标程序，它是计算机能直接识别和执行的程序。因此，用机器语言编写的程序能充分发挥其指令系统的特点，编出高质量的目标程序。但是，用机器语言编写的程序，其指令、数据和地址均以二进制数(十六进制数)表示，编程困难，不易读，不易交流，修改调试也很困难。

2. 汇编语言

为了克服机器语言编写程序的缺点，便于编写、调试、阅读和记忆，人们发明了用有助于记忆和理解的助记符来代表指令、数据和地址的汇编语言。汇编语言是一种面向机器的语言，它的助记符指令和机器语言保持着一一对应的关系。也就是说，汇编语言实际上就是机器语言的符号表示。前面所介绍的 80C51 系列 MCU 指令即属于汇编语言指令，而其指令代码即属于机器语言。

显然，用汇编语言编写的程序要比使用机器语言编写的程序简便、直观得多，而且使用汇编语言后，程序中的指令操作符、数据、地址分得较清楚，每条指令的意义也很明确。由于采用的是助记符，所以便于记忆，便于了解计算机的实际操作，也便于交流、修改和调试程序。因为它与机器的指令系统(机器语言)具有一一对应的关系，因此同样可以编写出高效率的程序。

3. 高级语言

高级语言是一种面向算法和过程的程序设计语言，采用更接近人类自然语言和习惯的数学表达式及直接命令的方法来描述算法和过程，如 BASIC 语言、C 语言等。高级语言的语句直观、功能强，一条语句往往相当于许多条指令，在程序设计时也不必顾及计算机的结构和指令系统，对不同的计算机，其程序基本能通用，尤其对复杂的科学计算和数据处理，高级语言有着明显的优势。

4. 三种语言比较

机器语言是计算机唯一能理解和执行的语言，用其编写的程序执行效率高、速度快，但由于不易编写、阅读、记忆、交流和推广，故其应用受到很大限制。

汇编语言是一种低级语言，是一种依赖机器的语言，因此具有机器语言编写程序的全部优点。用汇编语言可编制高质量的目标程序，即占用内存少，执行速度快，而且它又克服了机器语言的不易编写、调试、阅读、交流等缺点，尤其适用于实时应用场合的程序设计。但是，汇编语言仍不能独立于具体机器类型，即通用性不强。

高级语言易学易懂，通用性强，便于推广、交流。但是，用高级语言编写的程序经编译后所产生的目标程序质量相对较差，占用内存多，运行速度较慢，这在实时应用中是一个突出的问题。因此，高级语言一般用于科学计算和信息管理等。

3.4.2 汇编语言语句和格式

1. 汇编语言语句的种类

用汇编语言编写的程序称为汇编语言源程序。汇编语言源程序的基本组成单位称为汇编语言语句，故了解汇编语言程序的格式只要了解其语句结构即可。汇编语言语句有两种基本类型：指令语句、伪指令语句。



(1) 指令语句：每一条指令语句都在汇编时产生一个目标代码(机器码，machine code)，对应着机器的一种操作。

例如：MOV　　A, #0

(2) 伪指令语句：主要是为汇编语言汇编操作服务的语句，在汇编时不会形成目标代码。

例如：ONE　　EQU　　1

2. 汇编语言语句的格式

汇编语言的每个语句占有一行，典型的汇编语言语句由四个域组成：标号域，操作码域，操作数域及注释域。例如：

```
[标号：]    操作码    [操作数]    [；注释]
LOOP1:      MOV       R1, #32H    ；给 R1 赋值
```

每个语句必须具有操作码域，以说明这条语句的执行功能。操作数域可以是地址或数据，也可以空缺。标号域和注释域可有可无。为了使程序便于编写和阅读，可以给一个语句指定一个标号，还可以适当加上注释，对语句的作用进行说明。

1) 标号域

标号是为语句或某一程序段起的名字，用标号来标明某语句或某程序段第一条语句中指令代码第一个字节的地址，在程序的其他地方可以引用这个标号以代表这个特定的地址。因此，在一个完整的程序中，不同的语句或不同的程序段只能冠以不同的标号，即标号不能重复定义。规定标号由 1～8 个英文字母或数字组成(在 Keil C51 中，标号可由 127 个英文字母或数字组成)，但第一个符号必须是英文字母，并且必须以“：”结束。指令系统中的助记符、CPU 的寄存器名以及各伪指令均不能用作语句的标号。

一般情况下，只有那些被其他语句(如转移、调用)引用的语句和数据，才需要赋予标号。当无标号时，语句的标号域为空白。

2) 操作码域

操作码是每一个语句中不可缺少的部分，也是语句的核心部分。在这个域中书写 80C51 系列 MCU 的指令操作码或伪指令的助记符，如 MOV、ANL、CJNE、EQU 等。

3) 操作数域

操作数是指令的操作对象，可以是数据也可以是地址，根据操作要求，可以有 1～3 个操作数，也可以没有操作数。当操作数多于两个时，中间用“,”分开。出现在操作数域的数据和地址可归纳为如下几种。

(1) 常数。常数可以是参加运算的数，也可以是操作数的地址，这些数的表示法有多种。

① 二进制数：后缀为 B。例如：MOV　A, #00100001B。

② 十进制数：无后缀，也可将后缀设为 D。例如：MOV　A, #33 或 MOV　A, #33D。

③ 十六进制数：后缀为 H。例如：MOV　A, #21H。

要注意的是，以 A～F 开头的十六进制数，必须在其前面加上数字 0，如将立即数 A7H 送累加器 A，其相应指令的正确写法应该是 MOV　A，#0A7H。

(2) ASCII 码。当操作数为 ASCII 码字符时，应用“' '”把字符括起来，CPU 将对该字符的 ASCII 码值进行操作。

例如：MOV A, 'A'指令与 MOV A,#41H 指令等效(A 的 ASCII 码值为 41H)。

(3) 当前指令地址。用“$”表示程序计数器的内容，这是正在执行着的指令代码的第一字节地址。

例如：“SJMP $”指令表示循环执行该条指令。常用来等待中断的到来。

(4) 标号。标号也可作为指令的操作数。如：

```
LJMP    NEXT     ；跳转到以 NEXT 为起始地址的程序入口
LCALL   SUB      ；调用以 SUB 为起始地址的子程序
```

(5) 带有加减的表达式。例如：MOV A, SUM+1。

(6) 特殊功能寄存器名。例如：MOV A, P2。

4) 注释域

编写注释的目的是为了方便程序的阅读和交流，它是对程序段或语句功能的说明或解释，汇编时不产生任何指令代码。一般在程序的关键处或易混淆处加上简洁的文字注释。注释必须由“；”开始，即使内容换行也必须由“;”开始。

3.4.3 伪指令

汇编语言语句中的操作码域的操作码可以是指令的助记符，也可以是伪指令。对于汇编语言来说，伪指令仅是一种命令，用以控制汇编时执行一些特殊的操作，但这些命令并无对应的指令代码，汇编时也不产生目标程序代码，因而不影响程序的执行。由于它有指令的形式而无指令的实质，所以有“伪”指令之称。常用的伪指令有以下几条。

1. 定义起始地址伪指令(ORG)

格式：

```
ORG     操作数
```

此伪指令的操作数为一个 16 位的程序存储器地址。它指出其后的程序汇编成机器语言的目标程序后在程序存储器中存放的起始地址。如果程序中有多条 ORG 指令，则要求其操作数的值由小到大顺序安排，空间不允许重叠。例如：

```
地址       指令代码              源程序
                               ORG     1000H
1000H      755020     START:   MOV     50H, #20H
1003H      E550                MOV     A, 50H
1005H      2430                ADD     A, #30H
1007H      4037                JC      SWP
                               ORG     1040H
1040H      C4         SWP:     SWAP    A
```



第一条 ORG 伪指令是定义接下来的程序段的目标程序代码将从程序存储器地址 1000H 处开始存放，第二条 ORG 伪指令使后续指令(SWAP　A)从地址 1040H 开始存放目标程序代码。

2. 汇编结束伪指令(END)

格式：

```
[标号:] END [表达式]
```

这是汇编语言源程序的结束标志，放在源程序的末尾，源程序汇编为目标程序代码时，遇到 END 伪指令即结束对一个源程序的汇编。[ ]中的内容根据需要，可有可无。

例 3.24　把 40H 和 41H 两单元中的无符号数相加，结果存于 42H 单元中。

解：源程序如下。

```
        ORG     1000H
START:  MOV     A, 40H
        ADD     A, 41H
        MOV     42H, A
        END     START
```

3. 赋值伪指令(EQU)

格式：

```
标号        EQU        操作数
```

该伪指令的功能是使 EQU 两边的两个量相等，即标号值等于其操作数的值；其操作数可以是 8 位或 16 位的二进制数，也可以是事先定义的标号或表达式。例如：

```
VALUE       EQU     68H       ; VALUE=68H
EMPTY       EQU     1000H     ; EMPTY=1000H
BUF         EQU     EMPTY     ; BUF=1000H
```

需要注意的是，在某个程序中，一旦用 EQU 伪指令对某标号赋值之后，就不能再用 EQU 伪指令来改变其值。

4. 定义字节伪指令(DB)

格式：

```
[标号:] DB        表达式或表达式表
```

该伪指令的功能是把表达式或表达式表的数值存入本指令标号开始的一个存储单元或多个连续的存储单元中。其中的表达式是指一个数据字节或用单引号括起来的字符，表达式表是用逗号分开的多个字节或用双引号括起来的字符串，例如：

```
      地址          指令代码          源程序
                                  ORG      1100H
1100H       08                NUM:  DB       08H
1101H       FE 10 03 FC       BUF:  DB       -2, 10H, 3, -4
1105H       4C 6F 6E 67       STR:  DB       "Long"
```

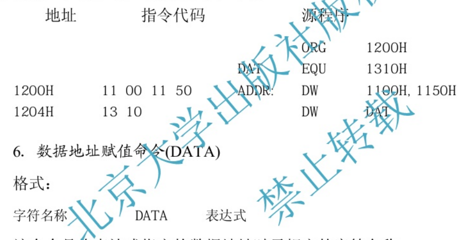
其中的字符以 ASCII 码形式存入程序存储器区，负数用补码形式存入程序存储器区。

5. 定义字伪指令(DW)

格式：

```
[标号:] DW        表达式或表达式表
```

DW 伪指令的功能与 DB 的功能相似，不同之处在于 DW 操作数中的表达式不是 8 位的数据字节，而是 16 位的数据字，表达式表中各表达式也用逗号隔开。在 Keil C 汇编时，DW 按高字节在前(低地址单元)低字节在后(高地址单元)的顺序处理。标号也可作为 DW 的操作数，但该标号必须事先赋值。例如：




```
      地址          指令代码          源程序
                                  ORG      1200H
                          DAT     EQU      1310H
1200H       11 00 11 50       ADDR: DW       1100H, 1150H
1204H       13 10                   DW       DAT
```

6. 数据地址赋值命令(DATA)

格式：

```
字符名称        DATA        表达式
```

该命令是将表达式指定的数据地址赋予规定的字符名称。

该伪指令的功能与 EQU 有些相似，区别如下：EQU 伪指令定义的标号必须是先定义后使用，而 DATA 伪指令则无此限制；用 EQU 伪指令可以把一个汇编符号赋给一个字符名称，而 DATA 伪指令则不能；DATA 伪指令可将一个表达式的值赋给一个字符变量，而 EQU 则不能这样使用。DATA 伪指令在程序中常用来给 SFR 定义字符名称。例如：

```
P0        DATA      80H
```

7. 定义存储空间命令(DS)

格式：

```
[标号:] DS        表达式
```

定义空间命令 DS 指定从某地址开始，保留若干字节空间备用。在汇编以后，将根据表达式的值来决定从指定地址开始留出多少个字节空间，表达式也可以是一个指定的数值。例如：



```
ORG        1300H
DS         10H
```

表示从 1300H 地址开始，保留 16 个连续的地址单元备用。

8. 位地址符号命令(BIT)

格式：

```
字符名称        BIT        位地址
```

该命令对位地址赋予所规定的字符名称。BIT 伪指令在程序中常用来给可位寻址 SFR 的位定义字符名称。例如：

```
CY              BIT        0D7H
AC              BIT        0D6H
```

这样就把两个位地址分别赋给两个字符 CY 和 AC，在编程中它们就可当作位地址来使用。

3.4.4　汇编方式

用汇编语言编写的源程序便于人们阅读和记忆，但是计算机并不能识别，必须将其转换为由二进制码组成的目标程序代码后计算机才能执行。将汇编语言的源程序转换为计算机所能识别的机器语言代码的目标程序的过程称为汇编。汇编又分手工汇编和计算机汇编。

1. 手工汇编

手工汇编指由汇编者对照指令表，将源程序的每条指令的指令代码分别查出，然后把这些指令代码以字节为单位从源程序的起始地址依次排列形成目标程序。

在汇编过程中，若碰到与后面程序有关的地址标号或变量，则暂时将这些单元空出，继续往后汇编，最后再根据后面的汇编结果，将这些空的单元填好。

2. 计算机汇编

汇编可由专门的程序来执行，这种程序称为汇编程序。由汇编程序来完成的汇编方式，称为计算机汇编。

由于计算机的发展和普及，以及计算机仿真技术的成熟，现在的汇编方式都是由汇编程序完成的。

3.5　汇编语言程序设计

3.5.1　汇编语言程序设计步骤

用汇编语言设计一个程序大致上可分为以下几个步骤。

1. 分析题意

解决问题之前，首先要明确所要解决的问题的要求，虽然这是不言自明的道理，但也是初学者最容易犯的错误之一。要细心地阅读和分析以文字形式表达的问题，不要急于编写程序。

2. 确定算法

根据实际问题的要求和指令系统的特点，决定所采用的计算公式和计算方法，这就是一般所说的算法，算法是进行程序设计的依据，决定了程序的正确性和程序的质量。

3. 设计程序流程图

解题步骤可以用一行行的文字来加以描述和说明，但当问题较复杂时，往往不容易用文字将问题描述得既正确又清楚，也不便于他人阅读。因此，人们常将文字步骤加以图解而成为流程图(程序框图)。

程序流程图是解题步骤及其算法进一步具体化的重要手段，是设计程序的重要依据。它能比较清楚、形象地表达程序运行的过程，并且直观、清晰地体现程序的设计思路，可以使人迅速抓住程序的基本线索。

流程图是由预先约定的各种图形、流程线及必要的文字符号构成的。

4. 编写源程序

设计程序流程图后，基本思路已比较清楚，接下来的任务就是编制源程序，也就是用程序设计语言把流程图所表达的步骤描述出来。如果选用汇编语言书写程序，则可用一些合适的汇编语言指令来实现流程图中每一框内的要求，从而编制出一个有序的指令流，这就是源程序设计。

5. 上机调试

只有通过上机调试并得出正确结果的程序，才能认为是正确的程序。对于 MCU 来说，没有自开发的功能，需要使用仿真器或利用仿真软件进行仿真调试，排除程序中的错误，直至正确为止。

6. 程序优化

程序优化的目的在于缩短程序的长度，加快运算速度和节省数据存储单元。例如，可恰当地使用循环程序和子程序结构，通过改进算法和正确使用指令来节省工作单元和减少程序执行的时间。

事实上，所谓程序设计方法，也只是在大量的程序设计实例中归纳、提炼出的某些规律而已。因此，读者要掌握程序设计的技术，除了多读懂一些程序实例，学习一些程序设计的方法之外，最重要的是自己应多练习设计各种各样的程序，并在计算机上调试，实现自己编制的程序。

3.5.2 顺序结构程序设计

顺序结构程序设计也称简单程序设计，是所有程序设计中最基本的一种，是程序设计



的基础。其主要特点是按操作顺序依次编程，程序中不包括分支、循环、子程序等，程序的走向是唯一的。在执行顺序程序时，完全按指令书写顺序从第一条指令开始执行，直到最后一条指令结束。

下面通过几个实例来说明顺序程序的结构和设计技术。

例 3.25　将片外数据存储器中 240H 的内容拆成两段，其高 4 位存入 241H 单元的低 4 位，其低 4 位存入 242H 单元的低 4 位。

解：源程序如下。

```
          ORG     0000H
          LJMP    MAIN
          ORG     0030H
MAIN:     MOV     SP, #60H
          MOV     DPTR, #240H
          MOVX    A, @DPTR      ; 取片外存储器中数送 A
          MOV     R0, A         ; 数据暂存于 R0
          SWAP    A             ; (A) 的高、低 4 位互换
          ANL     A, #0FH       ; 取原数的高 4 位, 屏蔽低 4 位
          INC     DPTR
          MOVX    @DPTR, A      ; 将高 4 位送 2041H 单元
          MOV     A, R0         ; 重新取数
          ANL     A, #0FH       ; 取原数的低 4 位
          INC     DPTR
          MOVX    @DPTR, A      ; 将低 4 位送 2042H 单元
          SJMP    $
          END
```

例 3.26　编写一个计算 $Y=X1+X2-X3$ 的程序。设 $X1$=38H，$X2$=2AH，$X3$=19H，Y 存放在内部 RAM 的 BUF 单元中。

解：源程序如下。

```
    X1    EQU     38H
    X2    EQU     2AH
    X3    EQU     19H
    BUF   EQU     30H
          ORG     0000H
          LJMP    MAIN
          ORG     0030H
MAIN:     MOV     SP, #60H
          MOV     A, #X1          ; X1 送 A
          ADD     A, #X2          ; X1+X2 送 A
          CLR     C
          SUBB    A, #X3          ; X1+X2-X3 送 A
          MOV     BUF, A          ; 结果送 BUF 单元
          SJMP    $
          END
```

例 3.27 利用查表法求函数值，计算 $Y=2X^2$，X 为 0～9 中的整数。

解：用查表法计算，应事先计算出 X=0～9 时相应的 Y 值，然后依次存入起始地址为 TAB 的程序存储器中，形成一个表。在计算 $Y=2X^2$ 时，只需按 X 的值求出 Y 值，即存放的程序存储器单元地址，就可求得其函数值。其地址计算法为表的起始地址 TAB 加 X 的值。

设 X 存放在 30H 中，Y 值存放在 31H 中，以 X=3 为例。源程序如下：

```
X       EQU     30H
Y       EQU     31H
_X_     EQU     3                       ; (X) =3
        ORG     0000H
        LJMP    MAIN
        ORG     0030H
MAIN:   MOV     SP, #60H
        MOV     X, #_X_
        MOV     DPTR, #TAB              ; 表首地址送 DPTR
        MOV     A, X                    ; A←(X)
        MOVC    A, @A+DPTR              ; 查表得到 2X² 的值, A←(2X²)
        MOV     Y, A                    ; Y←(A)
        SJMP    $
TAB:    DB      0, 2, 8, 18, 32, 50
        DB      72, 98, 128, 162
        END
```

程序流程图如图 3-5 所示。

开始
表首地址TAB送DPTR
30H单元中X的值送A
根据提供的偏移量查表
结果送31H单元
结束

图 3-5 查表法程序流程图



3.5.3　分支结构程序的设计

单纯由顺序结构构成的程序比较简单，应用有限。在实际问题中，往往需要计算机对某种情况做出判断，根据判断结果做出相应的处理。通常，计算机依据某些运算结果来判断和选择程序的不同走向，形成分支。因此，在形成分支时，一般要有测试、转向和标识三个部分。

(1) 测试。通过对程序状态寄存器 PSW 中各位状态的测试，或通过对指定的存储单元或指定的寄存器的某位(某些位或全部位)的测试，判断某条件是否成立，决定是否转移，形成分支。80C51 系列 MCU 指令系统中的条件转移类指令均具有这种测试功能，可用它们来实现。

(2) 转向。根据测试结果决定程序的走向。在源程序中由控制转移类指令完成，在流程图中以菱形逻辑框表示。

(3) 标识。对每个程序分支给出一个标识，以标明程序转移的方向。一般将分支程序转向的第一个语句赋予一个标号，作为此分支的标识。

需要注意的是，一条控制转移指令经一次判别只能形成两个分支。若要形成多分支程序，要由多个转移控制指令组合，经多次判别来实现。所以，分支程序的设计比顺序程序的设计复杂一些。

下面举例说明。

例 3.28　编制源程序，实现以下符号函数

$$Y=\begin{cases}1, & X>0;\\ 0, & X=0;\\ -1, & X<0;\end{cases}$$

解：设 X 存放在片内 RAM 的 40H 单元中，Y 存放在片内 RAM 的 41H 单元中，以 X=-5 为例，源程序如下：

```
X       EQU     30H
Y       EQU     31H
_X_     EQU     -5
        ORG     0000H
        LJMP    MAIN
        ORG     0030H
MAIN:   MOV     SP,#60H
        MOV     X,#_X_
        MOV     A,X               ;取数 A←(X)
        JZ      ZERO              ;若 X=0,转 ZERO
        JB      ACC.7,MINUS       ;若 X<0,转 MINUS
        MOV     Y,#1              ;若 X>0,则 Y←01H
        AJMP    ENDP
ZERO:   MOV     Y,#0              ;若 X=0,则 Y←00H
        AJMP    ENDP
```

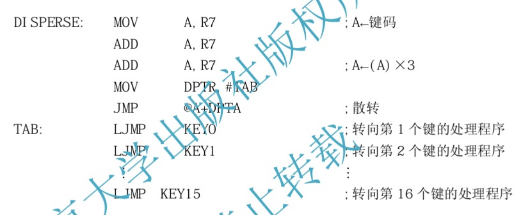
```
MINUS:    MOV    Y, #-1        ; 若 X<0, 则 Y←0FFH(Y←-1)
ENDP:     SJMP   $
          END
```

例 3.29 设计按键处理散转程序。

解：散转程序的功能是根据某一输入变量或运算结果的值，转向各个不同的处理程序入口地址。它是多路分支程序中的一种，80C51 系列 MCU 指令系统中的“JMP @A+DPTR”作为散转指令，可方便地实现多路分支。

MCU 应用系统一般有 16 个按键，经按键扫描程序得到某个按键的键码值(00H～0FH)存放在 R7 之中，16 个按键的按键处理程序入口地址分别为 KEY0，KEY1，…，KEY15。为此，可先在程序存储器中建立一张转移表，按 A 值从小到大的顺序从地址 TAB 开始，每三个单元写入 1 条相应的无条件转移指令，即 LJMP KEY0，LJMP KEY1，…，LJMP KEY15。

源程序(主要部分)如下：



```
DISPERSE:   MOV    A, R7          ; A←键码
            ADD    A, R7
            ADD    A, R7          ; A←(A) ×3
            MOV    DPTR, #TAB
            JMP    @A+DPTA        ; 散转
TAB:        LJMP   KEY0           ; 转向第 1 个键的处理程序
            LJMP   KEY1           ; 转向第 2 个键的处理程序
             ⋮                     ⋮
            LJMP   KEY15          ; 转向第 16 个键的处理程序
```


3.5.4 循环结构程序的设计

在上述顺序程序中，所有的指令只被执行一次，而在分支程序中有的指令被执行一次，有的可能一次也未被执行。事实上，一个复杂的计算问题，常常包括许多重复的计算步骤，这时可设法控制某段程序重复执行的次数，这种程序称为循环结构程序。使用循环结构程序的设计方法，能缩短程序设计的时间，节省程序的存储空间，所以循环结构程序设计是程序员必须掌握的一种程序设计技术。然而，要使程序能循环运行，需要一些附加的工作，如设置循环次数、设置结束判断指令等。因此，采用循环结构程序不能节省程序执行的时间。

1. 循环结构程序的基本结构

循环结构程序由以下四部分组成。

(1) 初始化部分。在进入循环结构程序主体之前做的必要的准备工作，给循环过程的工作单元设置初值。如循环控制计数初值的设置、地址指针的起始地址的设置、为变量预置初值等，都属于循环结构程序初始化部分。

(2) 处理部分。这是循环结构程序的核心部分，完成实际的处理工作，是需反复循环



执行的部分，故又称为循环体。这部分的程序内容，取决于实际需处理的问题。

(3) 循环控制部分。这是控制循环结构程序的循环与结束的部分，通过循环变量和结束条件进行控制。在重复执行循环体的过程中，不断修改循环变量，直到符合结束条件，就结束循环结构程序的执行。在循环过程中，除不断修改循环变量外，还需修改地址指针等有关参数。循环处理程序的结束条件不同，相应的循环控制部分的实现方法也不一样，分循环计数控制法和条件控制法。例如，计算结果达到给定的精度要求或找到某一个给定值时就结束循环等，这时的循环次数是不确定的。

(4) 结束部分。这部分是对循环结构程序执行的结果进行分析、处理和存放。

主机对循环结构程序的初始化和结束部分均只执行一次，而对循环体和循环控制部分则常需重复执行多次。循环体和循环控制部分是循环结构程序的主体，它影响着循环结构程序的效率，是循环结构程序设计的重点所在，应精心设计、正确编程。

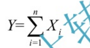
上述四部分有时能较明显地划分，有时则相互包含，不一定能明显区分。

2. 循环计数控制法

1) 单循环

单循环终止控制采用计数方法，即用一个寄存器作为循环次数计数器，每循环一次后加 1 或减 1，达到终止数值后循环停止。对于 80C51 系列 MCU，可以用减 1 不等于 0 转移控制指令 DJNZ 来实现计数方法的循环终止控制。工作寄存器 R0～R7 和片内数据 RAM 单元均可作为循环计数器，但 A 寄存器不能作为循环计数器。

例 3.30　编一段程序完成下列计算



$$Y=\sum_{i=1}^{n} X_i$$

设 n=10，X_i 顺序存放在片内 RAM 从 50H 开始的连续单元中，所求的和放在 R3 及 R4 中。

解：源程序如下：

```
CNT_ADDR    EQU     50H
CNT_N       EQU     10
            ORG     0000H
            LJMP    MAIN
            ORG     0030H
MAIN:       MOV     SP,#60H
            MOV     R2,#CNT_N           ;数组长度送 R2
            MOV     R3,#0               ;R3 清 0
            MOV     R4,#0               ;R4 清 0
            MOV     R0,#CNT_ADDR        ;数据区首地址送 R0
LOOP:       MOV     A,R4
            ADD     A,@R0
            MOV     R4,A                ;和数的低字节送 R4
            CLR     A
```

```
        ADDC    A, R3
        MOV     R3, A           ; 修改地址指针
        INC     R0
        DJNZ    R2, LOOP        ; 数据未加完，则继续执行
        SJMP    $
        END
```

该程序是用 R2 作为减法计数器的，同样也可用加法计数器来控制循环。

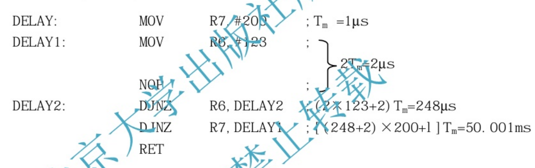
2) 多重循环

如前面介绍的，一个循环结构程序中不包含其他的循环结构程序，则称该循环结构程序为单循环结构程序；如果一个循环结构程序中包含了其他的循环结构程序，则称该循环结构程序为多重循环结构程序。这在实际问题中也是经常遇到的。

最简单的多重循环应用程序是由 DJNZ 指令构成的软件延时程序。

例 3.31 设计一个 50 ms 延时程序。

解：延时程序与指令的执行时间关系密切。在使用 12 MHz 晶振时，一个机器周期(T_m)为 1 μs，执行一条 DJNZ 指令需 2 个机器周期，即 2 μs。这时可用双重循环方法写出如下的延时 50 ms 程序(子程序)。



```
DELAY:    MOV     R7, #200        ; Tm =1μs
DELAY1:   MOV     R6, #123        ; ┐
                                  ; ├ 2Tm=2μs
          NOP                     ; ┘
DELAY2:   DJNZ    R6, DELAY2      ; (2×123+2) Tm=248μs
          DJNZ    R7, DELAY1      ; [(248+2) ×200+1 ] Tm=50.001ms
          RET
```

若需延时更长时间，可采用更多重的循环，如 1s 延时可用三重循环。

3. 条件控制法

以上介绍的循环程序的循环次数都是已知的，而有些问题其循环次数事先无法知道，但可以利用与问题有关的一些条件来控制循环。

例 3.32 把片内 RAM 中从 ST1 地址开始存放的数据区传输到 ST2 地址开始的存储区中，数据区长度未知，但已知数据区的最后一个字节内容为 00H，而其他字节均不为 0。设源地址空间与目的地址空间不重叠，ST1 地址为 40H，ST2 地址为 60H。

解：可利用判断每次传输的内容是否为 0 这一条件来控制循环。

源程序如下。

```
ST1     EQU     30H
ST2     EQU     50H
        ORG     0000H
        LJMP    MAIN
        ORG     0030H
```



```
MAIN:       MOV     SP,#70H
            MOV     R0,#ST1       ;源数据区首地址送 R0
            MOV     R1,#ST2       ;目标数据区首地址送 R1
LOOP:       MOV     A,@R0         ;取数据送累加器 A
            JZ      END_LOOP      ;若(A)=0 则跳出循环,返回
            MOV     @R1,A         ;若(A)≠0,将数据送目标数据区
            INC     R0            ;修改源数据区指针
            INC     R1            ;修改目标数据区指针
            SJMP    LOOP          ;继续执行
END_LOOP:   SJMP    $
            END
```

4. 循环结构程序设计中应注意的问题

(1) 循环结构程序是一个有始有终的整体，它的执行是有条件的，所以要避免从循环体外直接转移到循环体内部，因为这样做未经过初值设置，会引起程序的混乱。

(2) 多重循环结构程序是从外层向里层一层层进入，但要结束循环时，是由里层到外层一层层退出，所以在循环嵌套程序中，不要在外层循环中用转移指令直接转到里层循环体。

(3) 循环体内可以直接转到循环体外或外层循环中，实现一个循环由多个条件控制结束的结构。

(4) 在编写循环程序时，首先要确定程序的结构，弄清逻辑关系。一般来说，一个循环体的设计可以从第一次执行情况着手，先画出重复运算部分的流程图，然后加上修改、判断和设置初值部分，使其成为一个完整的循环结构程序。

5. 循环结构程序的优化

循环体是循环结构程序中重复执行的部分，如经过仔细推敲，合理安排，可使其执行时间缩短。故应对循环体进行优化，如从改进算法、选用最合适的指令和工作单元入手，可以达到缩短执行时间的要求。对于循环体来说，缩短程序的长度并不是特别重要，重要的是缩短程序的执行时间。

3.5.5　子程序设计

1. 子程序的概念

在实际的程序设计中，将那些需多次应用的、完成相同的某种基本运算或操作的程序段从整个程序中独立出来，单独编制成一个程序段，尽量使其标准化，并存放于某一存储区域，需要时通过指令进行调用。这样的程序段，称为子程序。调用子程序的程序称为主程序或调用程序。

使用子程序的过程称为子程序调用，可由专门的指令来实现，这种指令称为子程序调用指令(如 ACALL 或 LCALL)。

子程序执行完后，返回到原来程序的过程称为子程序返回，也由专门的指令来实现，这种指令称为子程序返回指令(RET)。

能供调用的子程序，必须具有以下两个特点:

(1) 子程序的第一条指令地址称子程序首地址或称入口地址，必须用标号标明，以便调用指令正确调用。

(2) 子程序的末尾用返回指令 RET 结束，以便正确返回主程序或调子程序继续执行。

2. 子程序的设计

子程序从结构上看，与一般程序相比没有多大的区别，唯一的一点是在子程序的末尾有一条子程序返回指令 RET，其功能是当子程序执行完后返回到主程序中去。为了能够正确地使用子程序，并在子程序执行完返回到主程序后又能正确地工作，在编写子程序时需要注意以下两点。

1) 参数传递

在调用子程序时，主程序应先将子程序所要用到的有关参数(即入口参数)放到某些约定的寄存器或存储单元中。子程序在运行时，可以从这些约定的地方得到有关参数。同样，子程序在运行结束前也应将运行结果(出口参数)送到约定的寄存器或存储单元中，以便返回主程序后，主程序可以从这些地方获得所需要的结果，这就是参数传递。

实现参数传递可以采用多种约定方法，下面按照 80C51 系列 MCU 的特点介绍几种常用的方法。

(1) 用工作寄存器或累加器传递参数。这种方法就是将入口参数或出口参数存放在工作寄存器或累加器中。使用这种方法所编写的程序最简单，运算速度最高。其缺点是工作寄存器数量有限，不能传递太多的数据；主程序必须先把数据送到工作寄存器；参数个数固定，不能由主程序任意设置。

(2) 用指针寄存器来传递参数。由于数据一般存放在存储器中，而不存放在工作寄存器中，所以可用地址指针来表示数据的位置，这样可以节省传递数据的工作量，并可实现可变长度计算。如果参数在片内数据 RAM 中，可用 R0 或 R1 做指针；如果参数在片外 RAM 中，可用 DPTR 做指针。进行可变长度运算时，可用一个寄存器来指出数据长度，也可在数据中指出其长度(如使用标记等)。

(3) 用堆栈传递参数。堆栈也可用于传递参数，调用时，主程序把参数压入堆栈中，以后子程序可根据堆栈指针间接访问堆栈中的参数，同时可把结果参数送回堆栈；返回主程序后，将结果参数弹出堆栈。这种方法的优点是简单、能传递大量参数、不必为特定的参数分配存储单元。中断服务程序将使用这种方法，由于参数在堆栈中，故可大大简化中断响应时的现场保护。

2) 保护现场与恢复现场

在子程序中，应按实际情况设置保护现场部分和恢复现场部分，其功能如下。

(1) 保护现场。在调用子程序时，由于程序转入子程序执行可能破坏主程序或调用程序的有关状态寄存器(PSW)、工作寄存器和累加器等的内容。因此，必要时应将这些寄存器单元内容保护起来，即保护现场。对于 PSW、A、B 等可通过压栈指令进栈保护。

(2) 恢复现场。当子程序执行完后，即返回主程序时，应先将上述内容送回来时的寄存器中，这一过程称为恢复现场。对于 PSW、A、B 等内容可通过出栈指令来恢复。

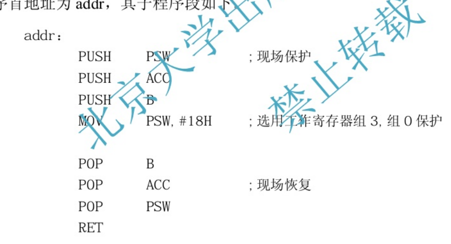


80C51 系列 MCU 应用系统，由于片内 RAM 容量小，限制了堆栈的深度。为了增强程序执行速度和实时性，工作寄存器不采用进栈保护的办法，而采用选择不同工作寄存器组的方式来达到保护的目的。一般主程序选用工作寄存器组 0，而子程序选用工作寄存器的其他组。这样既节省了入栈/出栈操作，又减少了堆栈空间的占用，且速度快。一般保护/恢复现场的方式有两种。

(1) 调用前保护，返回后恢复。这种方式是在主程序的调用指令之前进行现场保护；在调用指令之后，即返回断点后进行现场恢复。设子程序首地址为 addr，其主程序结构如下。

```
PUSH    PSW          ; 将 PSW、ACC、B 等压栈保护
PUSH    ACC
PUSH    B
MOV     PSW, #10H    ; 选用工作寄存器组 2, 将组 0 保护
ACALL   addr         ; 调用子程序
POP     B            ; 恢复 PSW、ACC、B 内容
POP     ACC
POP     PSW
```

(2) 调用后保护，返回前恢复。这种方式是在主程序调用后，在子程序的开始部分，进行必要的现场保护；而子程序结束，返回指令前进行现场恢复。这是常用方式，设子程序首地址为 addr，其子程序段如下。




```
addr:
        PUSH    PSW          ; 现场保护
        PUSH    ACC
        PUSH    B
        MOV     PSW, #18H    ; 选用工作寄存器组 3, 组 0 保护

        POP     B
        POP     ACC          ; 现场恢复
        POP     PSW
        RET
```

上述两种方式中，如果每次调用需保护的内容不同，可采用前者。但每次调用均需在主程序中编写保护和恢复程序，故会增加程序量，多占用存储空间。如果每次调用保护的内容固定，则应采用后者，这样，不仅减少程序量，且有利于程序的读、写、修改和调试。在编写子程序时，还应注意保护(压栈)和恢复(出栈)的顺序，即先压入者后弹出，否则将出错。

3. 子程序的特点

随着程序设计技术的发展，子程序的设计越来越重要。因此，对编制子程序应有较高要求，除通常在程序设计中应遵循的原则(程序应尽量简练、占用存储空间少、执行速度快等)外，还应具备以下特性。

1) 通用性

为使子程序能适应各种不同程序、不同条件下的调用，子程序应具有较强的通用性。例如，数制转换子程序、多字节运算子程序等，应能适应各种不同应用程序的调用。

2) 可浮动性

可浮动性是指子程序段可设置在存储器的任何地址区域。假如子程序段只能设置在固定的存储器地址段，这在编制主程序时要特别注意存储器地址空间的分配，防止两者重叠。为了能使子程序段浮动，必须在子程序中避免选用绝对转移地址，而应选用相对转移类指令，子程序首地址也应采用符号地址。

3) 可递归和可重入性

子程序能自己调用自己的性质，称为子程序的可递归性，而子程序能同时被多个任务(或多个用户程序)调用的性质，称为子程序的可重入性。这类子程序常在庞大而复杂的程序中应用，MCU 应用程序设计很少用到。

4) 子程序说明部分(说明文件)

对于通用子程序，为便于各种用户程序的选用，要求在子程序编制完成后提供一个说明部分(说明文件)，使用户不必详读子程序，只需阅读说明部分(说明文件)就能了解其功能及应用。子程序说明部分(说明文件)一般包含如下内容。

(1) 子程序名：标明子程序功能的名字。

(2) 子程序功能：简要说明子程序能完成的主要功能。

(3) 初始和结果条件：说明有哪些参量、参量传送和存储单元，说明执行结果及其存储单元。

(4) 所用的寄存器：提示主程序对哪些寄存器内容应进栈保护。

(5) 子程序调用：指明本子程序需调用哪些子程序。

有些复杂而庞大的子程序还需说明占用资源情况、程序算法及程序结构流程图等，随子程序功能的复杂程度不同，其说明部分(说明文件)的要求也各不相同。

例 3.33　将单字节无符号二进制整数转换成三位压缩型 BCD 码。

解：采用 80C51 的除法指令，可以很方便地实现单字节二进制整数转换成三位压缩型 BCD 码。三位 BCD 码需要占用 2 个字节，将百位 BCD 码存于高位地址字节单元，十位和个位 BCD 码存于低地址字节单元中。程序如下。

```
；子程序名：BINBCD。
；子程序功能：将单字节无符号二进制整数转换成三位压缩型 BCD 码。
；入口参数：8 位无符号二进制整数存于 R3 中。
；出口参数：三位 BCD 码存于 R4、R5 中。
；转换方法：采用除法指令。
BINBCD：
            PUSH      PSW            ；现场保护
            PUSH      ACC
            PUSH      B
```



```
        MOV     A, R3           ; 二进制整数送 A
        MOV     B, #100         ; 十进制数 100 送 B
        DIV     AB              ; A/100，以确定百位数
        MOV     R5, A           ; 商(百位数)存于 R5 中
        MOV     A, #10          ; 将 10 送 A 中
        XCH     A, B            ; 将 10 和 B 中余数互换
        DIV     AB              ; A/10 得十位、个位数
        SWAP    A               ; 将 A 中商(十位数)移入高 4 位
        ADD     A, B            ; 将 B 中余数(个位数)加到 A 中
        MOV     R4, A           ; 将十位、个位 BCD 码存入 R4 中
        POP     B
        POP     ACC             ; 恢复现场
        POP     PSW
        RET                     ; 返回
```

3.5.6　程序设计综合举例

1. 找最小值

例 3.34　试编写一个程序，找出从片外 RAM 的 0200H 单元开始的连续 10 个单元无符号数中的最小值，并将它存在 0300H 中。

解：源程序如下：



```
MIN:    MOV     DPTR, #0200H    ; 数据区首地址送 DPTR
        MOV     R0, #9          ; 循环次数送 R0
        MOVX    A, @DPTR
        MOV     30H, A          ; 取第一个数送 30H
LOOP:   INC     DPTR            ; 修改地址指针
        MOVX    A, @DPTR        ; 取下一个数送 A
        CJNE    A, 30H, NEXT
NEXT:   JNC     GREAT           ; 若(A)>(30H)，则转 GREAT
        MOV     30H, A          ; 取小值送 30H
GREAT:  DJNZ    R0, LOOP        ; 数据区未比较完，继续执行
        MOV     A, 30H
        MOV     DPTR, #0300H    ; 最小值送 0300H 单元
        MOVX    @DPTR, A        ;
        RET
```


2. 查找关键字

例 3.35　试编写一个程序，从片外 RAM 的 ST 单元开始的 100 个单元中查找某一数据 DT，若找到，将片内 RAM 的 BUF 单元清 0，并将该数据所在单元地址的高字节存入 BUF+1 单元，低字节存入 BUF+2 单元；若未找到，则将 0FFH 送入 BUF 单元。

解：源程序如下。

```
KEY:        MOV     DPTR, #ST        ; DPTR←数据区首地址
            MOV     30H, #DT         ; 30H←关键字
            MOV     R0, #100         ; 数据区长度送 R0
            MOV     R1, #BUF         ; 存储标志地址送 R1
LOOP:       MOVX    A, @DPTR         ; A←取数
            CJNE    A, 30H, NOT_KEY  ; 不是关键字，则转 NOT
            MOV     @R1, #0          ; 是关键字，则 BUF←00H
            INC     R1
            MOV     @R1, DPH         ; BUF+1←(DPH)
            INC     R1
            MOV     @R1, DPL         ; BUF+2←(DPL)
            AJMP    ENDP
NOT_KEY:    INC     DPTR             ; 修改地址指针
            DJNZ    R0, LOOP         ; 数据区未比较完，继续执行
            MOV     @R1, #0FFH       ; BUF←0FFH
ENDP:   RET
```


3. 无符号数排序

排序就是把某一数据区间连续存放的数据按其大小进行排列，并依次存放于一系列连续的存储单元中。

例 3.36 在片内数据存储器 ST 开始的单元中任意存放了 N 个无符号数，试编写一个程序，将这 N 个数在原数据区中由小到大进行排序。

解：

方法一：最小值法。

这种方法就是重复从数据区中找出最小数，并将这些最小数依次排列，第 1 次是找出 N 个数中的最小数，并换至数据区的第 1 个单元存放；第 2 次是在剩余的 $N-1$ 个数中找出最小数，并换到第 2 个单元存放；……，这样，经过 $N-1$ 次循环查找、存放，原数据区中便会得到由小到大顺序排列的数。

设 R7 存放外循环计数值，R6 存放内循环计数值。A 存放每次查找的最小值，30H 单元用作数据区首地址暂存，31H 单元用作内循环计数值暂存，R0 指向每次参与比较的数，R1 存放每次内循环找出的最小值的存放地址。

源程序如下。

```
SORT1:      MOV     R7, #N-1         ; 置外循环计数值
            MOV     R6, #N-1         ; 置内循环计数值
            MOV     R0, #ST          ; 置排序前数据区首地址
            MOV     R1, #ST          ; 置排序后数据区首地址
            MOV     30H, R0          ; 数据区首地址暂存 30H 单元
            MOV     31H, R6          ; 内循环计数值暂存 31H 单元
```



```
ROTA:     MOV     A, @R0          ; 取第一个数
LOOP:     INC     R0              ; 修改指针
          MOV     32H, @R0        ; ((R0))送 32H 单元
          CJNE    A, 32H, NEXT
NEXT:     JC      SKIP            ; 若(AX)<(32H), 则转 SKIP
          XCH     A, @R0          ; 若(A)>((R0)), 则两者交换
SKIP:     DJNZ    R6, LOOP        ; 若内循环未结束, 则继续执行
          MOV     @R1, A          ; 存最小数
          INC     R1              ; 修改指针
          MOV     R0, 30H
          INC     R0
          MOV     30H, R0         ; 修改数据区首地址
          MOV     R6, 31H
          DEC     R6
          MOV     31H, R6         ; 修改内循环计数值
          DJNZ    R7, ROTA        ; 若外循环未结束, 则继续执行
          MOV     A, @R0
          MOV     @R1, A          ; 存放数组中的最后一个数
          RET
```

方法二：冒泡法。

冒泡法的基本思路是：从数组的第一个数开始，相邻两单元的数相比较，如果第一个单元的数大，则两者交换。这样依次比较下去，重复上述过程 $N-1$ 次后，才能将 N 个数中的最大值沉降到数据区的最后一个单元中；然后又从头开始进行第二轮比较，经过 $N-2$ 次比较后，才能把第二个最大值沉降到数据区的倒数第二个单元；……，循环 $N-1$ 轮后，N 个数据便会由小到大顺序排列在原数据区中。使用标志位 F0 表示是否交换，设 R7 存放外循环计数值，R6 存放内循环计数值，A 存放后一个比较数，B 存放前一个比较数。源程序如下。

```
SORT2:                            ; 排序子程序
          MOV     R7, 40H         ; N 个数字，比较 N-1 次
S1:
          MOV     R0, 41H         ; 起始地址
          MOV     B, R7
          MOV     R6, B
          CLR     F0              ; 交换标志清零
S2:
          MOV     B, @R0          ; 取出前一个数
          INC     R0
          MOV     A, @R0          ; 取出后一个数
          CJNE    A, B, S3        ; 后－前
S3:
```

```
            JNC     GREAT           ；后一个数大就不用交换
            MOV     @R0, B          ；交换存放
            DEC     R0
            MOV     @R0, A
            INC     R0
            SETB    F0              ；设立交换标志位
GREAT:
            DJNZ    R6, S2
            JNB     F0, S_END       ；没有交换过，就结束
            DJNZ    R7, S1
            S_END:
            RET
```

本 章 小 结

指令是微处理器控制计算机进行某种操作的命令，而指令系统则是全部指令的集合，它是表征计算机性能的重要指标，每台计算机都有自己特有的指令系统。

用操作码表示的指令称为汇编语言指令，汇编语言源程序是由一条条汇编语言指令组成的。汇编语言指令格式如下。

[标号：] 操作码 [操作数 1][，操作数 2][，操作数 3][；注释]

其中，标号用于表示该条指令的地址，操作码用于表示指令进行何种操作，操作数包括源操作数和目的操作数。操作数是汇编指令格式中最复杂的部分，大致可分为 5 类：立即操作数#data、寄存器 Rn，存储单元(direct、@Ri、@A+PC)、位操作数 bit 和相对偏移量 rel。不同类型的操作数对应不同的寻址方式，80C51 单片机指令系统共有七种寻址方式，这七种寻址方式和寻址空间见表 3-1。

表 3-1　寻址方式和寻址空间对照表

序号	寻址方式	利用的变量	寻址空间
1	立即寻址	#data	程序存储器
2	直接寻址	direct	片内 RAM 低 128B 和 SFR
3	寄存器寻址	R0～R7,A,B,C,DPTR	工作寄存器和部分 SFR
4	寄存器间接寻址	@R0,@R1,@SP	片内 RAM
		@R0,@R1,@DPTR	片外 RAM 或 I/O 端口
5	变址寻址	@A+PC,@A+DPTR	程序存储器
6	相对寻址	PC+rel	程序存储器
7	位寻址	bit	片内 RAM 中位寻址区及可以位寻址的 SFR 位



80C51 单片机的指令按功能可分为传送、算术、逻辑、位处理和转移指令，现将前 4 种指令小结如下。

1. 传送指令

传送指令是完成片内数据存储器传送、片外数据存储器传送和程序存储器传送工作的指令。

片内数据传送指令为：MOV D,S;(D)←(S)

其中，目的操作数 D=A、n、direct 和@Ri，源操作数 S=#data、A、Rn、direct、@Ri。注意：寄存器 Rn 与 Rn 之间不能直接传送，寄存器 Rn 与存储单元@Ri 之间不能直接传送。

片外数据传送指令为：MOVX A，@DPTR 和 MOVX @DPTR，A。

程序存储器传送指令为：MOVC A，@A+DPTR 和 MOVX A，@A+PC。

交换指令为：XCH A，S 和 SWAP A 等。其中，S= Rn、direct、@Ri。

2. 算术运算指令

算术运算指令有加法、减法、乘法和除法，操作码分别为 ADD、SUBB、MUL 和 DIV。在算数运算指令中，目的操作数必须是累加器 A，而源操作数 S=#data、Rn、direct 和@Ri。因此在算术运算前必须将加数、被减数、被乘数、被除数送入累加器 A。

注意：在加法指令 ADD 之后加 DA A 指令可进行十进制加法运算，由于没有减法十进制调整指令，所以十进制减法运算必须用补码进行。

3. 逻辑运算指令

逻辑运算指令包括逻辑运算和移位指令。

(1) 逻辑运算。在双操作数的逻辑运算指令中累加器 A 总是作为目的操作数，而源操作数 S=#data、Rn、direct 和@Ri。与、或、异或指令的操作码分别为 ANL、ORL、XRL。

(2) 循环移位。A 循环左移指令为：RL A、A；带进位循环左移指令为：RLC A，A；循环右移指令为：RR A，A；带进位循环右移指令为：RRC A。

4. 位处理指令

位处理指令分为位传送、位修改和位逻辑三类。

位传送指令：MOV C，bit 和 MOV bit，C。

位置 1 和清 0 指令：SETB bit 和 CLR bit。

位逻辑指令包括：与指令 ANL C，bit 和 ANL C，/bit；或指令 ORL C，bit 和 ORL C，/bit。

思考题与习题

1. 简述 80C51 系列 MCU 的寻址方式及所涉及的寻址空间。
2. 变址寻址方式有什么优点？主要用于什么场合？
3. 访问 SFR 和片外 RAM 应采用哪种寻址方式？
4. 对 80C51 系列 MCU 片内数据区地址 80H～0FFH 的空间寻址时应注意些什么？
5. 80C51 系列 MCU 的指令系统具有哪些主要特点？

6．80C51 系列 MCU 有哪些逻辑运算功能？各有什么用处？

7．80C51 系列 MCU 的转移类指令有何独特优点？无条件转移指令有哪几种？如何选用？

8．80C51 系列 MCU 的短调用和长调用指令本质上有何区别？如何选用？

9．80C51 系列 MCU 的片内 RAM 中，已知(30H)=38H，(38H)=40H，(40H)=48H，(48H)=90H。分析下面各条指令，说明源操作数的寻址方式，给出按顺序执行各条指令后的结果。

```
MOV     A, 40H
MOV     R0, A
MOV     P1, #0F0H
MOV     @R0, 30H
MOV     DPTR, #3848H
MOV     40H, 38H
MOV     R0, 30H
MOV     0D0H, R0
MOV     18H, #30H
MOV     A, @R0
MOV     P2, P1
```

10．已知(A)=7AH，(R0)=30H，(30H)=0A5H，(PSW)=80H，请填写下列各条指令的执行结果：

```
(1)     SUBB    A, 30H
(2)     SUBB    A, #30H
(3)     ADD     A, R0
(4)     ADD     A, 30H
(5)     ADD     A, #  30H
(6)     ADDC    A, 30H
(7)     SWAP    A
(8)     XCHD    A, @R0
(9)     XCH     A, R0
(10)    XCH     A, 30H
(11)    XCH     A, @R0
(12)    MOV     A, @R0
```

11．试分析以下程序段的执行结果。

```
MOV     SP, #3AH
MOV     A, #20H
MOV     B, #30H
PUSH    ACC
PUSH    B
POP     ACC
POP     B
```



12．已知(A)=81H，(R0)=17H，(17H)=35H，指出执行完下列程序段后 A 的内容。

```
ANL     A, #17H
OR1     17H, A
XR1     A, @RO
CPL     A
```

13．设 R0 的内容为 32H，A 的内容为 48H，内部 RAM 的 32H 单元内容为 80H，40H 单元内容为 08H，指出在执行下列程序段后上述各单元内容的变化。

```
MOV     A, @RO
MOV     @RO, 40H
MOV     40H, A
MOV     RO, #35H
```

14．将片外 RAM 区 100CH 单元中的内容传送到 120CH 单元中，请编程实现。

15．将片外 RAM 区 40H 单元中内容和 41H 单元中内容相乘，并将结果存放在片外 RAM 区 42H 和 43H 单元中，高位存放在高地址中，请编程实现。

16．将片外 RAM 区 40H～60H 区域的数据块，全部搬移到片内 RAM 区的相同地址区域，并将原数据区全部填 00H，请编程实现。

17．计算片内 RAM 区 50H～57H 区域的数据的算术平均值，结果存放在 58H 中，请编程实现。

18．已知 16 位二进制数放在片内 RAM 区 20H 和 21H 单元，高位存放在高地址中，请编写将其右移一位的程序。

19．请用位操作指令，求下列逻辑方程：

(1) $P1.5 = ACC.0 \cdot (B.0 + P3.0) + \overline{P3.1}$

(2) $PSW.5 = P1.0 \cdot \overline{ACC.2} + B.5 \cdot \overline{P1.5}$

20．已知 16 位二进制数放在 R7R6 中，请编写对它们进行求补操作的程序，结果存放在 R1R0 中。

21．在起始地址为 1200H，长度为 64 的数据表中找出 ASCII 码“F”，将其送到 1000H 单元中，请编程实现。

22．试编写一段程序，把 0500H～0506H 单元中的压缩 BCD 码转换成 ASCII 码，存放在 0500H 为首地址的存储单元中。

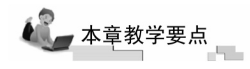
23．请编写一个延时 2ms 的子程序。

24．利用查表技术将累加器中的一位 BCD 码转换为相应的十进制数的七段码，结果仍放在 A 中(设显示数字 0～9 的七段码分别是：40H，79H，24H，30H，19H，12H，02H，78H，00H，IBH)。

25．为什么 SJMP 指令的 rel=0FEH 时，将实现单指令的无限循环？

第 4 章

80C51 系列微控制器的功能单元



知识要点	掌握程度	相关知识
并行 I/O 接口	了解并行 I/O 接口的性质、功能； 掌握并行 I/O 接口的编程和使用	P0 口、P1 口、P2 口、P3 口； 并行 I/O 接口功能及应用
定时器/计数器	了解定时器/计数器的概念； 熟悉定时器/计数器的工作方式； 掌握定时器/计数器的编程和使用	定时器/计数器 T0、T1； 定时器/计数器 T2； 定时器/计数器的编程和使用
中断响应过程及中断应用编程	了解中断系统概念； 熟悉中断操作； 掌握中断的编程和使用	中断过程及其控制和操作； 外部中断源扩展； 中断的编程和使用
串行口的工作方式及应用编程	了解串行口的结构及其工作方式； 掌握串行口的编程和应用	特殊功能寄存器； 工作方式和多机通信方式； 波特率发生器和波特率； 编程和应用



导入案例

智能手机发展历史——功能的集成

智能手机是指比固定电话以及只能使用短信服务(SMS)的传统手机具备更高功能的手机，多为可管理备忘录、计算器及日程表等的PDA(个人数字信息助理终端)与手机相互融合的产品。

智能手机的发展史大致可分为 1990 年智能手机刚刚问世的“黎明期”、2000—2006 年商务智能手机繁荣发展的“商用机扩大期”以及 2007 年以后逐步走进普通消费者视野的“大众普及期”。

1. 智能手机黎明期(20 世纪 90 年代)

全球首款智能手机是美国 IBM 公司 1994 年投放市场的“IBM Simon”。这款手机装配备了使用手写笔的触摸屏，除了通话功能之外，还具备 PDA 及游戏功能。OS 采用的是夏普 PDA 的“Zaurus OS”。



IBM Simon　　　　诺基亚 9110i

1996 年芬兰诺基亚公司推出了名为“Nokia 9000 Communicator”的折叠式智能手机。该产品在折叠状态下就是一款手机，打开后则会出现 QWERTY 键盘、十字键及长方形黑白显示屏等。OS 采用美国 Breadbox Computer Company 的“GEOS”。Nokia 9000 Communicator 受到了商务人士的青睐，后来逐步演变为 1998 年上市的“诺基亚 9110”、由诺基亚 9110 按照美国的手机频率改进而来并于 2000 年上市的“诺基亚 9110i”，后来又推出了采用 Symbian OS 的机型。

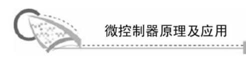
1997 年，瑞典爱立信公司推出了与 Nokia 9000 Communicator 相似的“GS88”手机。该手机的说明书中首次出现了“智能手机”一词。

2. 商用机扩大期(2000—2006 年)

进入 2000 年以后，市场上出现了很多采用面向 PDA 及嵌入式设备的通用 OS 智能手机。这些手机使用 Symbian、Palm OS 及 Windows CE 等 OS。

首次采用 Symbian OS 的智能手机是爱立信的“Ericsson R380 Smartphone”。该机的数字键部分采用可像门一样开关的机构，打开后会出现长方形触摸屏，可作为 PDA 使用。合上后可作为手机使用。继爱立信之后，诺基亚也于 2000 年投放了采用 Symbian OS 的智能手机，后来诺基亚的智能手机便一直使用 Symbian OS。

至于 Windows CE 智能手机系统，最早是美国微软公司 2002 年发布的“Microsoft Windows Powered Smartphone 2002”。配备该系统的首款手机是微软自己推出的“Orange SPV”(由台湾宏达国际电子生产)。该 OS 系列后来被更名为“Windows Mobile”，韩国三星电子及夏普等公司向市场投放了多款采用这种 OS 的智能手机。



加拿大 RIM(Research In Motion)公司的现行“黑莓”(BlackBerry)手机的首款原始机型问世于 2003 年。该机配备 QWERTY 键，融合了电子邮件、SMS 及 Web 浏览等功能。



Ericsson R380 Smartphone



“黑莓”(BlackBerry)手机

以上这些手机均以企业用户为目标，以嵌入商务软件的形式提供，因此基本未向普通消费者推广。

3. 大众普及期(2007 年至今)

美国苹果公司于 2007 年 6 月投放市场的 iPhone 掀起了让普通消费者购买并使用智能手机的潮流。这款手机配备了几乎所有操作都以触摸屏完成的用户界面(UI)、基本与个人电脑同等的 Web 浏览器和电子邮件功能，以及与 iTunes 联动的音乐播放软件等，从而将智能手机提高到了任何人都能使用的水平。

随后，美国谷歌公司于 2007 年 11 月发布了智能手机软件平台 Android。2008 年，美国 T-Mobile USA 公司推出了首款配备 Android 的智能手机“T-Mobile G1”(由台湾宏达国际电子生产)。此后，美国摩托罗拉移动公司、三星电子及日本与瑞典的合资公司索尼爱立信移动通信等公司都相继推出了 Android 智能手机。

之前一直开发企业用智能手机OS的微软在看到 iPhone 与 Android 成功之后也转变了方针，于 2009 年 2 月宣布开发出面向普通消费者的“Windows Mobile 6.5”及“Windows Phone 7”。采用 Windows Mobile 6.5 的手机于 2009 年 10 月投放市场，Windows Phone 7 手机则于 2010 年 10 月问世。



点燃智能手机普及之火的第一代 iPhone



首款配备 Android 的智能手机“T-Mobile G1”



Windows Phone 7

4. 当前主流智能手机

iOS 系统：iOS(又称 iPhone OS)是由苹果公司为 iPhone 开发的操作系统，它主要是给 iPhone、iPod touch 及 iPad 使用。



iOS 代表机型 iPhone5S



Android 代表机型 NEXUS5



WP 代表机型诺基亚 Lumia1020





Android 系统：Android (安卓) 是基于 Linux 平台的操作系统，越来越受到玩家的青睐，支持厂商很多。目前市面上几大系统中，Android 的市场占有率最高，上升速度最快。

Windows Phone 系统：作为软件巨头微软的掌上版本操作系统，在与桌面 PC 和 Office 办公的兼容性方面具有先天的优势。以商务用机为主，目前市场已显出颓势，最新版本为 2013 年与诺基亚联合发布的 Windows Phone 8 操作系统。

智能手机除了具备手机的基本功能，能够进行正常的通话及发短信等手机应用外，还具备以下功能：

(1) 具备无线接入互联网的能力，即需要支持 GSM 网络下的 GPRS 或者 CDMA 网络下的 CDMA 1X 或者 3G 网络。

(2) 具备 PDA 的功能，包括 PIM(个人信息管理)、日程记事、任务安排、多媒体应用、浏览网页。

(3) 具有开放性的操作系统，在这个操作系统平台上，可以安装更多的应用程序，从而使智能手机的功能得到无限扩充。

智能手机的发展就是各种功能集成的过程。同样，对于微控制器来说，在芯片内部集成越来越多的功能部件是其发展的方向。基本的 80C51 系列微控制器芯片内部集成了 4 类功能部件，而 Silicon Lab 公司的 C8051F 系列微控制器则集成了绝大部分的功能部件，使用这些微控制器设计的电子产品小巧可靠。

4.1 并行 I/O 接口

4.1.1 I/O 接口概述

I/O 接口是 CPU 和外围设备之间交换信息的连接部件，是 CPU 与外设之间进行数据传送的桥梁和纽带。外围设备种类繁多，有机械式、机电式或电子式等，有输入设备、输出设备，外围设备的工作速度通常比 CPU 的速度低很多，且不同外围设备的工作速度、信息类型、传送方式差别很大，因此导致 CPU 与外设之间的信息传送十分复杂，所以 CPU 和外设之间必须有接口电路，通过接口电路协调单片机与外设之间的数据传送。

1. I/O 接口的功能

在单片机应用中，输入设备通过 I/O 接口电路把程序、数据或现场采集到的各种信息输入单片机，单片机的处理结果和控制信息要通过 I/O 接口电路传送到输出装置，以便显示、打印或实现各种控制。一般来说 I/O 接口电路的主要功能如下。

1) 地址译码

由译码器对地址进行译码，选择外围设备，以便 CPU 对被寻址的外设进行读/写操作。

2) 数据缓冲和锁存

CPU 通过总线与多个外设打交道。但是，各输入设备的数据线不能都直接与 CPU 的数据总线相连，必须经输入缓冲器接到数据总线上；否则，会出现几个输入设备同时占用数据总线，发生“总线冲突”，以致 CPU 不能正常工作。缓冲电路便于实现在同一时刻

CPU 只与一个外设交换信息，即只有被选中的外设与 CPU 交换信息。

单片机传送信息的速度一般远远高于外设的工作速度。当单片机向外设输出信息时，外设还来不及立即将信息处理完毕。例如，点阵式打印机打印一个字符约需 10ms，而单片机输出一个字符只需 10μs 左右。因此，在输出接口电路中应设置数据锁存器，以便及时把 CPU 输出的数据锁存起来，然后再由外设进行处理。

3) 信息转换

外设送往单片机的信息应该转换成单片机所能接收的数字量，而单片机输出的信息应该转换成外设所要求的信号。因此，I/O 接口电路应能实现信息的转换。例如，串行、并行数据的互相转换，电压、电流的转换，电平的转换，模/数的转换，数/模的转换等。

4) 通信联络

为了协调 CPU 与外设之间的信息交换，CPU 需要通过 I/O 接口电路以一定的方式与外设进行通信联络，以保证不丢失信息。为了进行通信联络，I/O 接口电路传送的信息除了数据之外，还要提供外设状态信息及 CPU 对外设的启停控制信号等。

2. 接口与端口

I/O 接口的功能主要通过电路实现，因此也称之为接口电路。在接口电路中应该包含数据寄存器以保存输入输出数据，状态寄存器以保存外设的状态信息，命令寄存器以保存来自 CPU 的有关数据传送的控制命令。由于在数据的传送中，CPU 需要对这些寄存器的状态口和保存命令的命令口寻址，因此通常把接口电路中这些已编址并能进行读或写操作的寄存器称为端口(Port)，或简称口。因此，一个接口电路就对应着多个端口地址，对它们像存储单元一样进行编址。

3. 80C51 系列微控制器端口

标准 80C51 系列微控制器共有 4 个 8 位的并行双向口，有 32 根输入/输出(I/O)口线。各口的每一位均由锁存器、输出驱动器和输入缓冲器所组成。因为它们在结构上存在一些差异，所以各口的性质和功能也就有了差异。下面分别介绍 P0～P3 这 4 个端口。

4.1.2 P0 口

P0 口是一个多功能的 8 位端口，可以字节访问也可以位访问，其字节访问地址为 80H，位访问地址为 80H～87H。P0 口的各位结构完全相同，但相互之间又是独立的。P0 口的某一位的位电路结构图如图 4-1 所示，其中 i 的取值范围为 0～7。

1. 位电路结构

从图 4-1 可以看出，P0.i 的电路包含以下内容：

(1) 一个数据输出锁存器，用于输出数据锁存。

(2) 两个三态缓冲器 BUF1、BUF2，分别用于缓冲锁存器数据和引脚数据；

(3) 一个 2 选 1 的数据选择器 MUX21，其中一个输入来自锁存器的 $\overline{Q}$ 端，另一个输入为“地址/数据”信号的反相输入，选择控制端为“控制”。

(4) 数据输出的驱动电路和控制电路，由两个场效应管 T1、T2 和一个与门组成。





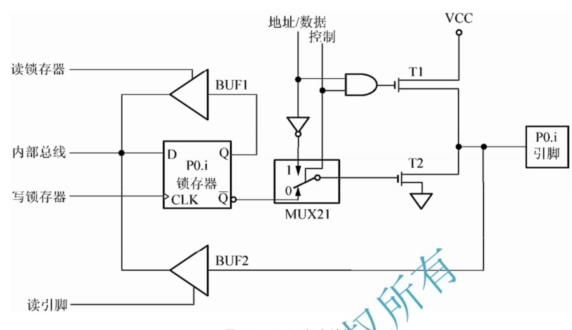


图 4-1　P0.i 电路结构

2. 工作过程分析

1) P0 口用作“地址/数据”总线

(1) 当 MUX21 的选择控制端置 1 时，选择“地址/数据”输出到 T2，此时 P0 口输出地址/数据。

当 MUX21 的选择控制端“控制”置 1 时，“地址/数据”信号控制场效应管 T1，而“地址/数据”信号的反相信号控制场效应管 T2。当输出的地址/数据信号为 1 时，与门输出为 1，场效应管 T1 导通，场效应管 T2 截止，P0.i 引脚输出为 1；当输出的地址/数据信号为 0 时，与门输出为 0，场效应管 T1 截止，场效应管 T2 导通，P0.i 引脚输出为 0。

通过上述分析可以看出，此时的输出状态随地址/数据线变化；而且，场效应管 T1、T2 轮流导通，构成了推拉式的输出电路，其负载能力大大增加。场效应管 T1 起到内部上拉电阻的作用。

(2) 当 MUX21 的选择控制端置 0 时，从 P0 口输入数据。

当 MUX21 的选择控制端“控制”置 0 时，选择 $\overline{Q}$ 输出到 T2。由于 P0 口作为地址/数据复用方式访问外部存储器时，CPU 自动向 P0 口写入 FFH(即此时 P0.i 锁存器的 $\overline{Q}$=1)，使场效应管 T2 截止，场效应管 T1 由于“控制”信号为 0 也截止，从而使输出处于高阻抗状态，保证外部存储器输入的数据直接由 P0.i 引脚通过输入缓冲器 BUF2 进入内部总线。

真正的双向口是具有高电平、低电平和高阻抗 3 种状态的端口。显然，此时 P0 口在不输入数据时处于高阻状态，因此，P0 口作为地址/数据总线使用时是一个真正的双向端口，简称双向口。

2) P0 口用作通用 I/O 口

当系统不进行外部程序存储器和数据存储器扩展时，即不将 P0 口用作系统的地址/数据总线使用时，P0 口可作为通用的 I/O 口使用。

当 P0 口用作通用 I/O 口时，MUX21 的选择控制端“控制”置 0，选择$\overline{Q}$端输出到 T2，同时与门输出为 0，场效应管 T1 截止，则 P0 口的输出电路为漏极开路输出，形成 OD 门。

(1) P0 口输出数据。当 P0 口输出数据时，来自 CPU 的“写”脉冲加到锁存器的 CLK 端，内部总线上的数据写入锁存器，并通过 T2 输出到 P0.i 引脚。当锁存器内的数据为 1 时，则$\overline{Q}$输出端为 0，场效应管 T2 截止，由于输出为漏极开路，此时必须外接上拉电阻才能输出 1(高电平)。当锁存器内的数据为 0 时，场效应管 T2 导通，P0 口输出为 0(低电平)。

(2) 从 P0 口输入数据。当从 P0 口输入数据时，有两种读入方式：“读引脚”和“读锁存器”(对应两种从 P0 口读入数据指令)。

当 CPU 发出“读引脚”指令时，锁存器的输出必须为 1(即$\overline{Q}$端输出为 0)，从而使场效应管 T2 截止，引脚的状态经三态缓冲器 BUF2 进入内部总线；反之，若“读引脚”时锁存器的输出为 0(即$\overline{Q}$端输出为 1)，则场效应管 T2 导通，从而使得 P0.i 引脚电平被钳位在 0(低电平)，使输入值 1(高电平)无法读入。此外，在场效应管 T2 导通的状态下，P0.i 引脚的高电平被强行拉回为低电平，从而可能产生很大的输入电流，将场效应管 T2 烧毁。

当 CPU 发出“读锁存器”指令时，锁存器内的数据通过 Q 输出端经由三态缓冲器 BUF1 进入 CPU 内部总线。之所以会有“读锁存器”的指令(例如“读-改-写”指令 ANL P0,A)，是因为“读锁存器”可以避免一些错误的读操作。例如，用 P0.i 引脚去驱动晶体管的基极。当对端口 P0.i 写入值 1(高电平)时，引脚 P0.i 为高电平(已外接上拉电阻)，则晶体管导通，该引脚被钳位在低电平。若此时 CPU 立刻去读该引脚的值，则由于晶体管导通，读入 CPU 的值为 0。显然这个值是错误的，因为锁存器输出的值为 1。

3. P0 口的特点

综上所述，P0 口具有以下功能和特点。

(1) 作为地址/数据复用总线使用。此时，P0 相当于一个真正的双向口，用作与外部存储器的连接，配合 P2 口，输出 16 位地址的低 8 位地址和输入/输出 8 位数据。当 P0 口作地址/数据复用总线用之后，就再也不能作 I/O 口使用了。当然，由于现在生产的多种 80C51 单片机内部集成了足够的 ROM 和 RAM，所以，P0 口很少用作地址/数据复用总线，大部分情况，P0 口仅用作通用 I/O 口。

(2) 作为 I/O 口使用。此时，P0 口需要外接上拉电阻，端口只有两个状态：高电平、低电平，是一个准双向口。为了能够从引脚读入正确的数据，当 P0 口由输出状态转变为输入状态时，应首先向锁存器 P0.i 写 1(执行指令 MOV P0,#0FFH)。当单片机复位后，锁存器 P0.i 被自动置 1，可立即作输出口使用。

4.1.3 P1 口

P1 口只有一个功能：I/O 口。对 P1 口来说，可以字节访问也可以位访问，其字节访



问地址为 90H，位访问地址为 90H～97H。P1 口的各位结构完全相同，但相互之间又是独立的。P1 口的某一位的位电路结构图如图 4-2 所示，其中 i 的取值范围为 0～7。

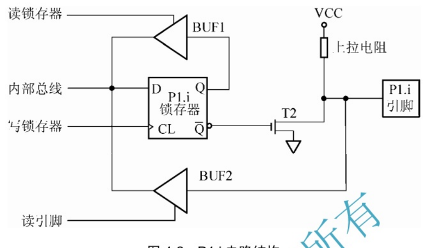


图 4-2　P1.i 电路结构

1. 位电路结构

从图 4-2 可以看出，P1.i 的电路包含以下内容：

(1) 一个数据输出锁存器，用于输出数据锁存。

(2) 两个三态缓冲器 BUF1、BUF2，分别用于缓冲锁存器数据和引脚数据。

(3) 数据输出的驱动电路，由一个场效应管 T2 和一个片内上拉电阻组成。

2. 工作过程分析

P1 口只能作为通用 I/O 口使用。

(1) P1 口用作输出口。CPU 输出 0 时，D=0，$\overline{Q}$=1，场效应管 T2 导通，P1.i 引脚被下拉为低电平，即输出 O；CPU 输出 1 时，D=1，$\overline{Q}$=0，场效应管 T2 截止，P1.i 引脚被上拉为高电平，即输出 1。

(2) P1 口用作输入口，此时分为“读引脚”和“读锁存器”两种方式。“读引脚”时，与 P0 口类似，先向 P1.i 位写 1，使场效应管 T2 截止，P1.i 引脚上的输入数据经输入缓冲器 BUF2 送入内部总线。“读锁存器”时，P1.i 锁存器的输出端 Q 的状态经输入缓冲器 BUF1 送入内部总线。

3. P1 口的特点

P1 口由于有片内上拉电阻，在作为输出口时，不需要在片外接上拉电阻；而且由于没有高阻抗状态，所以 P1 口是一个准双向口。

为了能够从引脚读入正确的数据，当 P1 口由输出状态转变为输入状态时，应首先向锁存器 P1.i 写 1(执行指令 MOV P1,#0FFH)。当单片机复位后，P1.i 锁存器被自动置 1，可立即作输出口使用。

4.1.4 P2 口

P2 口是一个多功能的 8 位端口，可以字节访问也可以位访问，其字节访问地址为 A0H，位访问地址为 A0H～A7H。P2 口的各位结构完全相同，但相互之间又是独立的。P2 口的某一位的位电路结构图如图 4-3 所示，其中 i 的取值范围为 0～7。

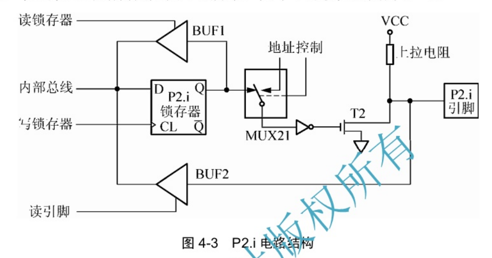


图 4-3 P2.i 电路结构

1. 位电路结构

从图 4-3 可以看出，P2.i 的电路包含以下内容：

(1) 一个数据输出锁存器，用于输出数据锁存。

(2) 两个三态缓冲器 BUF1、BUF2，分别用于缓冲锁存器数据和引脚数据。

(3) 一个 2 选 1 的数据选择器 MUX21，其中一个输入来自锁存器的 Q 端，另一个输入为“地址”信号，选择控制端为“控制”。

(4) 数据输出的驱动电路和控制电路，由一个场效应管 T2 和一个片内上拉电阻组成。

2. 工作过程分析

P2 口有两个功能：用作地址总线和通用 I/O 口。

(1) P2 口用作输出 16 位地址总线的高 8 位地址。在内部“控制”信号作用下，MAX21 选择“地址”输出到反相器的输入端。当“地址”为 0 时，场效应管 T2 栅极信号为 1，T2 导通，P2.i 引脚输出 0；当“地址”为 1 时，场效应管 T2 截止，P2.i 引脚输出为 1。

(2) P1 口用作通用 I/O 口。在内部“控制”信号作用下，MAX21 选择 Q 输出到反相器的输入端。

当 CPU 输出 0 时，Q＝0，T2 导通，P2.i 引脚输出 0；当 CPU 输出 1 时，Q＝1，场效应管 T2 截止，P2.i 引脚输出 1。

当 P2 口用作输入口时，此时分为“读引脚”和“读锁存器”两种方式。“读引脚”时，与 P0 口类似，先向 P2.i 位写 1，使场效应管 T2 截止，P2.i 引脚上的输入数据经输入缓冲器 BUF2 送入内部总线。“读锁存器”时，P2.i 锁存器的输出端 Q 的状态经输入缓冲器 BUF1 送入内部总线。



3. P2 口的特点

当 P2 口作为地址输出线使用时，P2 口输出 16 位地址线的高 8 位地址，与 P0 口输出的低 8 位地址一起构成 16 位地址，可以寻址 64KB 的地址空间。当 P2 口作为高 8 位地址输出时，输出锁存器 P2.i 的内容保持不变。

当 P2 作为通用 I/O 口使用时，P2 口为一个准双向口，功能与 P1 口相同。

4.1.5　P3 口

P3 口是一个多功能的 8 位端口，可以字节访问也可以位访问，其字节访问地址为 B0H，位访问地址为 B0H～B7H。P3 口的各位结构完全相同，但相互之间又是独立的。P3 口的某一位的位电路结构图如图 4-4 所示，其中 i 的取值范围为 0～7。

1. 位电路结构

从图 4-4 可以看出，P3.i 的电路包含以下内容：

(1) 一个数据输出锁存器，用于输出数据锁存。

(2) 3 个三态缓冲器 BUF1、BUF2 和 BUF3，分别用于缓冲锁存器数据、引脚数据和第二功能数据。

(3) 数据输出的驱动电路和控制电路，由一个“与非门”、一个场效应管 T2 和一个片内上拉电阻组成。

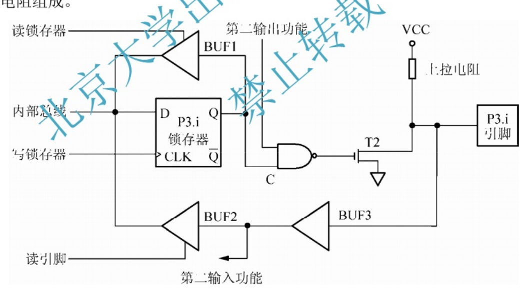


图 4-4　P3.i 电路结构

2. 工作过程分析

P3 口有两个功能：通用 I/O 口(第一功能)和第二输入/输出功能。

(1) P3 口用作通用 I/O 口。

当 P3 口用作第一功能通用输出时，“第二输出功能”端应置 1，使“与非门”处于开启状态，输出由 Q 端决定。当 CPU 输出 0 时，Q=0，场效应管 T2 导通，P3.i 引脚输出为

0；当 CPU 输出 1 时，Q=1，场效应管 T2 截止，P3.i 引脚输出为 1。

当 P3 口用作第一功能通用输入时，为了使场效应管 T2 截止，不影响输入电平，P3.i 锁存器和“第二输出功能”均应置 1。此时，在“读引脚”信号的作用下，P3.i 引脚数据通过三态缓冲器 BUF3、BUF2 进入内部总线。

与其他口类似，在“读锁存器”信号的作用下，P2.i 锁存器的输出端 Q 的状态经输入缓冲器 BUF1 送入内部总线。

(2) P3 口用作第二输入/输出功能。

与第一功能类似，当选择第二输出功能时，P3.i 锁存器置 1，使“与非门”处于开启状态，输出由“第二输出功能”端决定。当“第二输出功能”端输出为 0 时，场效应管 T2 导通，P3.i 引脚输出为 0；当“第二输出功能”端输出为 1 时，场效应管 T2 截止，P3.i 引脚输出为 1。

与第一功能类似，当选择第二输入功能时，P3.i 锁存器和“第二输出功能”均应置 1，保证场效应管 T2 处于截止状态。此时，P3.i 引脚输入的“第二功能”信息由输入缓冲器 BUF3 的输出送至内部引线“第二输入功能”。

3. P3 口的特点

P3 口片内有上拉电阻，无论是第一功能——通用 I/O 口，还是第二功能，皆为准双向口。

无论 P3 口用作通用 I/O 口的输入口，还是用作第二功能的输入/输出口，都需要将 P3.i 锁存器置 1。在实际应用中，由于复位后 P3.i 锁存器的初值为 1，所以，P3 口可以直接用作第二功能的输入/输出口而不需要做任何设置。

由于 P3 口各个位相互独立，当某一位不作为第二功能使用时，可以作为通用 I/O 口使用。

P3 口的第二功能定义如下：

P3.0——RXD，串行输入口。

P3.1——TXD，串行输出口。

P3.2——$\overline{\text{INT0}}$，外部中断 0 的请求。

P3.3——$\overline{\text{INT1}}$，外部中断 1 的请求。

P3.4——T0，定时器/计数器 0 外部计数脉冲输入。

P3.5——T1，定时器/计数器 1 外部计数脉冲输入。

P3.6——$\overline{\text{WR}}$，外部数据存储器写选通，输出，低电平有效。

P3.7——$\overline{\text{RD}}$，外部数据存储器读选通，输出，低电平有效。

4.1.6 并行 I/O 接口的编程和使用

例 4.1 P1 口输入/输出的简单应用，按键控制 LED 的点亮和熄灭。电路图如图 4-5 所示。

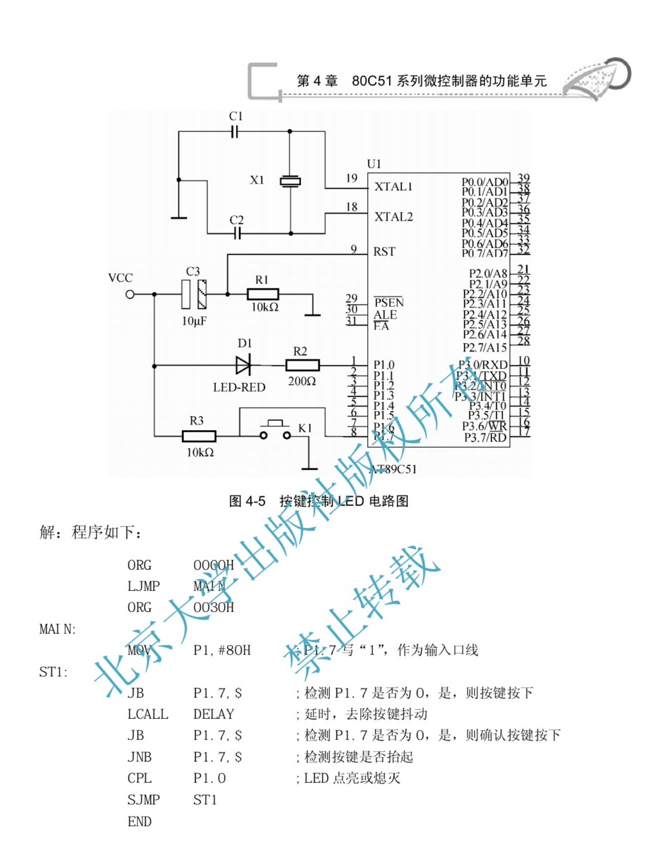


图 4-5　按键控制 LED 电路图

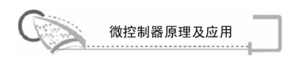
解：程序如下：

```
          ORG       0000H
          LJMP      MAIN
          ORG       0030H
MAIN:
          MOV       P1, #80H          ; P1.7 写“1”，作为输入口线
ST1:
          JB        P1.7, $           ; 检测 P1.7 是否为 0，是，则按键按下
          LCALL     DELAY             ; 延时，去除按键抖动
          JB        P1.7, $           ; 检测 P1.7 是否为 0，是，则确认按键按下
          JNB       P1.7, $           ; 检测按键是否抬起
          CPL       P1.0              ; LED 点亮或熄灭
          SJMP      ST1
          END
```


4.2　定时器/计数器

4.2.1　定时器/计数器概述

定时器/计数器(Timer/Counter)是单片机中重要的功能部件之一，其工作方式灵活、编程简单，对减轻 CPU 的负担和简化外围工作电路有重要意义。以 STC89C51 系列单片机



为例，单片机的定时器/计数器有以下特点：

(1) STC89C51 单片机包含有 2 个 16 位的定时器/计数器：定时器/计数器 T0 和定时器/计数器 T1。

(2) STC89C52 单片机包含有 3 个 16 位的定时器/计数器：定时器/计数器 T0、定时器/计数器 T1 和定时器/计数器 T2。

(3) 定时器/计数器的核心是一个加 1 计数器，其基本功能是加 1 功能。在单片机的 T0、T1 或 T2 引脚上施加一个 1 到 0 的跳变，计数器加 1，即为计数功能；在单片机内部对机器周期或其分频进行计数，计数值与周期的乘积，即为定时时间。

在 80C51 系列单片机中，定时功能和计数功能的设定和控制都是通过软件来进行的。

4.2.2 定时器/计数 T0、T1

1. 定时器/计数器 T0、T1 的内部结构

定时器/计数器 T0、T1 的内部结构简图如图 4-6 所示。从图中可以看出，定时器/计数器 T0、T1 由以下几部分组成：

(1) 计数器 TH0、TL0 和 TH1、TL1；

(2) 特殊功能寄存器 TMOD 和 TCON；

(3) 时钟分频器；

(4) 输入引脚 T0、T1、$\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$。

图 4-6　定时器/计时器 T0、T1 的内部结构简图

2. 定时器/计数器 T0、T1 的特殊功能寄存器

1) 定时器/计数器 T0、T1 的方式寄存器(TMOD)

顾名思义，80C51 系列单片机定时器/计数器的方式寄存器 TMOD 就是用来选择定时



器/计数器的工作方式的。方式寄存器 TMOD 是一个逐位定义的 8 位寄存器，是只能字节寻址的寄存器，字节地址为 89H，其格式如图 4-7 所示。

D7	D6	D5	D4	D3	D2	D1	D0
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
T1				T0			

图 4-7　方式寄存器 TMOD 格式

8 位被分作了两组，高 4 位定义定时器/计数器 T1 的工作方式，低 4 位定义定时器/计数器 T0 的工作方式。下面对 TMOD 的各位加以说明。

(1) GATE 门控位。

GATE=0 时，仅由运行控制位 TRi(i=0,1)来控制定时器/计数器计数。

GATE=1 时，由外部中断引脚 $\overline{INT0}$、$\overline{INT1}$ 和 TRi 来控制定时器/计数器计数。当 $\overline{INT0}$ 引脚为高电平时，TR0 置位，启动定时器/计数器 T0 计数；当 $\overline{INT1}$ 引脚为高电平时，TR1 置位来启动定时器/计数器 T1 计数。

(2) C/$\overline{T}$——计数功能、定时功能选择位。

C/$\overline{T}$ = 0 时，选择定时功能，定时器/计数器对单片机的晶振 12 分频后得到的信号进行计数，以此得到定时的时间。

C/$\overline{T}$ = 1 时，选择计数功能，定时器/计数器对外部输入引脚 T0(P3.4)或 T1(P3.5)的输入脉冲(负跳变)计数。

(3) M1、M0——工作方式选择位。由于有 M1 和 M0 两位，所以有四种工作方式见表 4-1。

表 4-1　定时器/计数器 T0、T1 的四种工作方式

M1	M0	工作方式	定时器/计数器配置功能
0	0	方式 0	13 位定时器/计数器
0	1	方式 1	16 位定时器/计数器
1	0	方式 2	自动重装载的 8 位定时器/计数器
1	1	方式 3	仅适用于 T0，T0 分为两个 8 位定时器/计数器，T1 停止计数

2) 定时器/计数器 T0、T1 的控制寄存器(TCON)

80C51 系列单片机定时器/计数器的控制寄存器 TCON 是用来控制、指示定时器/计数器的工作状态的。控制寄存器 TCON 是一个逐位定义的 8 位寄存器，既可字节寻址，也可位寻址。字节地址为 88H，位寻址的地址为 88H～8FH，其格式如图 4-8 所示。

位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
位功能	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

图 4-8　控制寄存器 TCON 格式

下面对 TCON 的各位加以说明。

(1) TF1、TF0(TCON.7、TCON.5) ——定时器/计数器 T1、T0 的溢出标志。

当定时器/计数器溢出时，该位由内部硬件置位。若中断开放，则响应中断，进入中断服务程序后，由硬件自动清零；若中断禁止，则此位可用于查询方式，作为状态位供 CPU 查询，当查询有效时，进入处理程序后，及时用软件将此位清 0。

(2) TR1、TR0(TCON.6、TCON.4) ——定时器/计数器 T1、T0 的运行控制位。

用软件控制，置 1 时，启动 T1；清零时，停止 T1。

TCON 的低 4 位与外部中断有关，将在后续章节中详细讨论。

复位后，TCON 的所有位均清零。

3) 定时器/计数器 T0、T1 的数据寄存器(TH1、TL1 和 TH0、TL0)

定时器/计数器 T0、T1 各有 1 个 16 位的数据寄存器，它们都是由高 8 位寄存器和低 8 位寄存器所组成的。这些寄存器不经过缓冲，直接显示当前的计数值。这 4 个寄存器都是读/写寄存器，任何时候都可对它们进行读/写操作。复位后，所有这 4 个寄存器全部清零。

它们都只能字节寻址，相应的字节地址见表 4-2。

表 4-2 定时器/计数器 T0、T1 的数据寄存器的字节地址

寄存器	名　称	字节地址
TH1	T1 的高 8 位数据寄存器	8DH
TL1	T1 的低 8 位数据寄存器	8BH
TH0	T0 的高 8 位数据寄存器	8CH
TL0	T0 的低 8 位数据寄存器	8AH

3. 定时器/计数器 T0、T1 的定时、计数模式选择

定时器/计数器 T0、T1 的工作模式是通过 TMOD 中的位 $C/\overline{T}$ 来选择的。

1) 定时器($C/\overline{T}=0$)

此时，计数输入信号是内部时钟脉冲，每个机器周期使寄存器的值加 1。每个机器周期包含 12 个振荡周期，故计数速率为振荡周期的 1/12。当采用 12MHz 的晶体时，计数速率为 1MHz。

定时器的定时时间与系统的振荡频率有关，与计数器的长度和初值有关。

2) 计数器($C/\overline{T}=1$)

这时，通过引脚 T0(P3.4)和 T1(P3.5)对外部信号进行计数。

在每个机器周期，CPU 都采样引脚的输入电平。若前一机器周期采样值为 1，下一机器周期采样值为 0，则计数器加 1，此后的机器周期，新的计数值装入计数器，周而复始，直至计数器溢出。由此可知，在计数模式下，检测到一个 1 到 0 的跳变(下降沿)需要 2 个机器周期，故最高计数频率为振荡频率的 1/24。



4. 定时器/计数器 T0、T1 的 4 种工作方式

通过设置 M1 和 M0 的值，定时器/计数器 T0、T1 可选择 4 种不同的工作方式。下面分别介绍。

1) 方式 0

当 TMOD 中的 M1M0=00 时，定时器/计数器被设置为工作方式 0。这时定时器/计数器的逻辑结构框图如图 4-9 所示。这种方式下，计数寄存器由 13 位组成，即 TLi 的高 3 位未用。

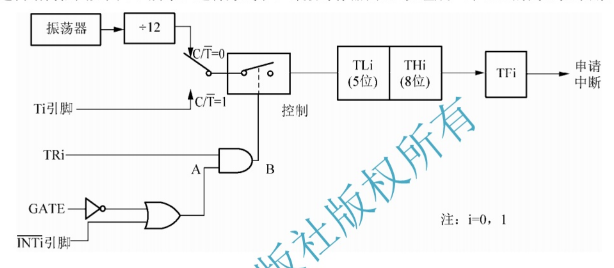


图 4-9　定时器/计数器方式 0 逻辑结构框图

计数时，TLi 的低 5 位溢出后向 THi 进位，THi 溢出后使 TFi 置位，并向 CPU 申请中断。如果中断允许，CPU 响应中断并转入中断服务程序，由内部硬件清 TFi。TFi 也可以由程序查询和清零。

是否计数，由 GATE、$\overline{INTi}$ 引脚、TRi 三部分决定，一般先设定 GATE 位。

(1) 当 GATE=0 时，A 点为高电平，定时器/计数器的启动/停止由 TRi 决定。TRi=1，定时器/计数器启动；TRi=0，定时器/计数器停止。

(2) 当 GATE=1 时，A 点的电位由 $\overline{INTi}$ 决定，因而 B 点的电位由 TRi 和 $\overline{INTi}$ 决定，即定时器/计数器的启动/停止由 TRi 和 $\overline{INTi}$ 两个条件决定。

2) 方式 1

当 TMOD 中的 M1M0=01 时，定时器/计数器被设置为工作方式 1。这时定时器/计数器的逻辑结构框图如图 4-10 所示。这种方式下，计数寄存器由 16 位组成。

计数时，TLi 溢出后向 THi 进位，THi 溢出后使 TFi 置位，并向 CPU 申请中断。如果中断允许，CPU 响应中断并转入中断服务程序，由内部硬件清 TFi。TFi 也可以由程序查询和清 0。其他与方式 0 完全相同。

3) 方式 2

当 TMOD 中的 M1M0=10 时，定时器/计数器被设置为工作方式 2。这时定时器/计数器的逻辑结构框图如图 4-11 所示。这种方式将 16 位计数寄存器分为 2 个 8 位寄存器，组成一个自动重装载的 8 位计数寄存器。

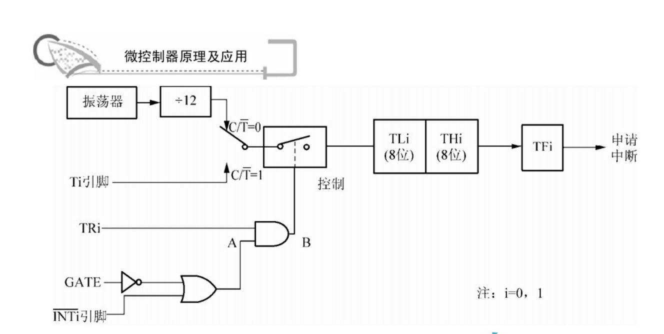


图 4-10　定时器/计数器方式 1 逻辑结构框图

在方式 2 中，TLi 作为 8 位计数寄存器，THi 作为 8 位计数常数寄存器。

当 TLi 计数溢出时，一方面将 TFi 置位，并向 CPU 申请中断；另一方面，将 THi 的内容自动加载到 TLi 中，继续计数。

重装载不影响 THi 的内容，因而可以多次连续加载。

方式 2 对定时控制特别有用，它可实现每隔预定时间发出控制信号，而且特别适合于串行口波特率发生器的使用。

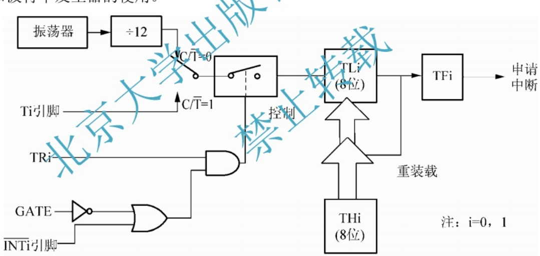


图 4-11　定时器/计数器方式 2 逻辑结构框图

4) 方式 3

当 TMOD 中的 M1M0=11 时，定时器/计数器被设置为工作方式 3。这种方式将定时器/计数器 T0 分为一个 8 位定时器/计数器和一个 8 位定时器，TL0 用于 8 位定时器/计数器，TH0 用于 8 位定时器。这时定时器/计数器 T0 的逻辑结构框图如图 4-12 所示。

定时器/计数器方式 3 下定时器/计数器 T0 的工作方式与方式 0、1 时相同，只是此时的计数器为 8 位计数寄存器 TL0，它占用了定时器/计数器 T0 的 GATE、$\overline{INT0}$、TR0、T0 引脚以及中断源等。由于定时器/计数器 T0 的资源已被计数寄存器 TL0 所占用，所以 TH0 只能作为定时器用，而且 TH0 占用了定时器/计数器 T1 的启动/停止控制位 TR1、计数溢出标志位 TF1 及中断源。

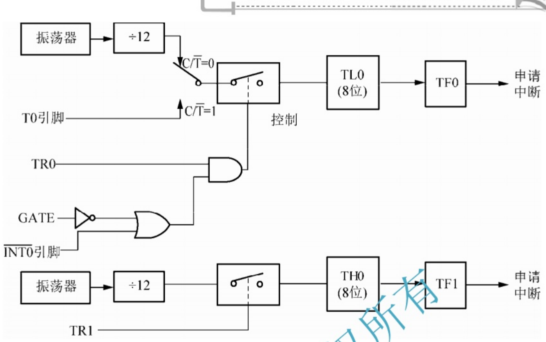


图 4-12　定时器/计数器方式 3 下定时器/计数器 T0 的逻辑结构框图

在定时器/计数器方式 3 下，定时器/计数器 T1 的结构如图 4-13 所示。此时定时器/计数器 T1 可选方式为 0、1 或 2。因为此时中断源已被占用，所以仅能作为波特率发生器或用在其他不用中断的地方。

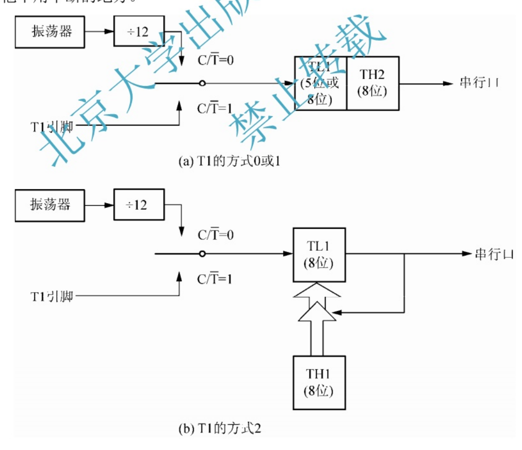


图 4-13　定时器/计数器方式 3 下定时器/计数器 T1 的逻辑结构框图

事实上，只在定时器/计数器 T1 用作波特率发生器时，定时器/计数器 T0 才选作方式 3。

4.2.3 定时器/计数器 T2

80C52 中有一个功能强大的定时器/计数器 T2，它是一个 16 位的、具有自动重装载和捕获能力的定时器/计数器。在定时器/计数器 T2 的内部，除了两个 8 位计数器 TL2、TH2 和控制寄存器 T2CON、方式寄存器 T2MOD 之外，还设置有捕获寄存器 RCAP2L(低字节)和 RCAP2H(高字节)。定时器/计数器 T2 的计数脉冲源可以有两个：一个是内部机器周期，另一个是由 T2(P1.0)端输入的外部计数脉冲。

输入引脚 T2(P1.0)是外部计数脉冲输入端。输入引脚 T2EX(P1.1) 是外部控制信号输入端。

1. 定时器/计数器 T2 的特殊功能寄存器

1) 定时器/计数器 T2 的控制寄存器(T2CON)

控制寄存器 T2CON 是一个逐位定义的 8 位寄存器，既可字节寻址，也可位寻址。字节地址为 0C8H，位寻址的地址为 0C8H～0CFH，其格式如图 4-14 所示。

位地址	OCFH	OCEH	OCDH	OCCH	OCBH	OCAH	OC9H	OC8H
位功能	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T2}$	$CP/\overline{RL2}$

图 4-14 控制寄存器 T2CON 格式

下面对 T2CON 的各位加以说明。

(1) TF2 ——定时器/计数器 T2 溢出标志。定时器/计数器 T2 溢出时置位，并申请中断。只能软件清除此标志位。当 RCLK=1 或 TCLK=1 时，即 T2 工作在波特率发生器方式下时，定时器/计数器 T2 的溢出不影响 TF2，即 TF2 不置 1。

(2) EXF2——定时器/计数器 T2 外部标志。当 EXEN2=1，且 T2EX 引脚上出现下降沿而造成捕获或重装载时，EXF2 置位，并申请中断。此时若已允许定时器/计数器 T2 中断，则 CPU 将响应中断，转向中断服务程序。只能软件清除此标志位。

(3) RCLK ——接收时钟标志位。此标志位由软件置位或清零，用以选择定时器/计数器 T2 或 T1 作串行口接收波特率发生器。RCLK=1 时，用定时器/计数器 T2 溢出脉冲作为串行口的接收时钟；RCLK=0 时，用定时器/计数器 T1 的溢出脉冲作接收时钟。

(4) TCLK ——发送时钟标志位。此标志位由软件置位或清零，用以选择定时器/计数器 T2 或 T1 作串行口发送波特率发生器。TCLK=1 时，用定时器/计数器 T2 溢出脉冲作为串行口的发送时钟；TCLK=0 时，用定时器/计数器 T1 的溢出脉冲作为串行口的发送时钟。

(5) EXEN2 ——定时器/计数器 T2 外部允许标志。此标志位由软件置位或清零。当 EXEN2=1 时，允许用外部信号来触发捕获或重装载操作；若定时器/计数器 T2 未用作串行口的波特率发生器，当在 T2EX 端出现下降沿信号时，将导致定时器/计数器 T2 捕获或重装载，并置 EXF2 标志为 1，请求中断。当 EXEN2=0 时，禁止用外部信号来触发捕获或重装载操作，即 T2EX 端的外部信号不起作用。

(6) TR2 ——定时器/计数器 T2 运行控制位。此标志位由软件置位或清零，以决定定时



器/计数器 T2 是否运行。TR2=l，启动定时器/计数器 T2；TR2=0，则停止定时器/计数器 T2。

(7) $C/\overline{T2}$——定时器/计数器 T2 的定时器方式或计数器方式选择位。此标志位由软件置位或清零。$C/\overline{T2}=0$，选择定时器工作方式；$C/\overline{T2}=1$，选择计数器工作方式，由下降沿触发计数。

(8) $CP/\overline{RL2}$——捕获/重装载标志。此标志位由软件置位或清零。$CP/\overline{RL2}=1$，选择捕获功能，这时当 EXEN2=1 且 T2EX 端出现信号下降沿时，发生捕获操作。$CP/\overline{RL2}=0$，选择重装载功能，这时若定时器/计数器 T2 溢出或 EXEN2=1，且 T2EX 端出现下降沿信号，都会导致自动重装载操作。当 RCLK+TCLK=l 时，$CP/\overline{RL2}$ 控制位不起作用，定时器/计数器 T2 被强制工作于重装载方式。重装载发生于定时器/计数器 T2 溢出时，常用作波特率发生器。

2) 定时器/计数器 T2 的方式寄存器(T2MOD)

方式寄存器 T2MOD 的字节地址为 0C9H，不可位寻址，其格式如图 4-15 所示。

D7	D6	D5	D4	D3	D2	D1	D0
—	—	—	—	—	—	T2OE	DCEN

图 4-15　方式寄存器 T2MOD 格式

虽然方式寄存器 T2MOD 有 8 位，但只定义了 2 位，其余位保留，且复位值均为 0。下面对 T2MOD 定义的 2 位加以说明。

(1) T2OE：定时/计数器 T2 输出允许控制位。

当 T2OE=1 时，启动定时器/计数器 T2 的可编程时钟输出功能，允许时钟输出至引脚 T2(P1.0)；当 T2OE=0 时，禁止引脚 T2(P1.0)输出。

(2) DCEN：定时/计数器 T2 加减计数控制位。

当 DCEN=1 时，允许 T2 作为加/减计数器使用。具体的计数方向由 T2EX 引脚来控制，当 T2EX=1 时，T2 进行加计数；当 T2EX=0 时，T2 进行减计数。DCEN=0 时，T2 自动加计数。

3) 定时器/计数器 T2 的数据寄存器(TH2、TL2)

定时器/计数器 T2 的数据寄存器是一个 16 位寄存器，它由高 8 位寄存器(TH2)和低 8 位寄存器(TL2)所组成，相应的字节地址为 OCDH 和 OCCH，只能字节寻址。这两个寄存器都是读/写寄存器。复位后，这 2 个寄存器全部清 0。

4) 定时器/计数器 T2 的捕获寄存器(RCAP2H、RCAP2L)

定时器/计数器 T2 中的捕获寄存器是一个 16 位寄存器，由高 8 位寄存器(RCAP2H)和低 8 位寄存器 (RCAP2L)所组成，相应的字节地址为 OCBH 和 OCAH，只能字节寻址。RCAP2H、RCAP2L 用于捕获计数器 TL2、TH2 的计数状态，或用来预置计数初值。TH2、TL2 和 RCAP2H、RCAP2L 之间接有双向缓冲器(三态门)，用于捕获或者重装载。复位后，这两个寄存器全部清零。

2. 定时器/计数器 T2 的定时、计数模式选择

定时器/计数器 T2 的工作模式是通过 T2CON 中的位 $C/\overline{T2}$ 来选择的。

1) 定时器($C/\overline{T2}=0$)

此时，T2 与 T0、T1 相似，计数输入信号是内部时钟脉冲，计数速率为振荡周期的 1/12，即每个机器周期使(TH2、TL2)寄存器的值加 1。

2) 计数器($C/\overline{T2}=1$)

这时，通过引脚 T2(P1.0)对外部信号进行计数，T2 的工作情况和时序关系与 T0、T1 完全一样，对外部计数脉冲的要求也相同，即外部脉冲的最高频率为振荡频率的 1/24。当有外部脉冲到来时，(TH2、TL2)寄存器的值加 1。

3. 定时器/计数器 T2 的 3 种工作方式

通过设置定时器/计数器 T2 的控制寄存器 T2CON 中的 $CP/\overline{RL2}$ 和 RCLK+TCLK 的值，定时器/计数器 T2 可选择 3 种不同的工作方式(表 4-3)：捕获方式、自动重装载方式和波特率发生器方式。

表 4-3 定时器/计数器 T2 的三种工作方式

RCLK+TCLK	$CP/\overline{RL2}$	TR2	工作方式
0	0	1	自动重装载方式
0	1	1	捕获方式
1	×	1	波特率发生器方式
×	×	0	关闭

下面分别介绍。

1) 自动重装载方式

自动重装载方式是指在一定条件下，自动地将 RCAP2H 和 RCAP2L 的数据装入计数器 TH2 和 TL2 中。一般说来，RCAP2H 和 RCAP2L 在这里起预置计数初值的功能。对于 89C52，其工作的逻辑结构框图如图 4-16 所示。

当 $CP/\overline{RL2}=0$ 时，选择自动重装载方式。重装载操作在下述两种情况下发生：

(1) 当定时器/计数器 T2 的寄存器(TH2、TL2)溢出时，打开重装载三态缓冲器，把(RCAP2H、RCAP2L)的内容自动装载到(TH2、TL2)中(重装载操作)。同时，溢出标志 TF2 置 1，申请中断。

(2) 当 EXEN2=1 且端口 T2EX(P1.1)出现下降沿信号时，将发生重装载操作。同时，标志位 EXF2 置 1，申请中断。

若定时器/计数器 T2 的中断是被允许的，则无论发生 TF2=1 还是 EXF2=1，CPU 都会响应中断，此中断向量的地址为 2BH。响应中断后，应采用软件方式撤除中断申请，以正确地响应新的中断。TF2 和 EXF2 都是直接可寻址位，可采用指令“CLR TF2”和“CLR EXF2”撤除中断申请。

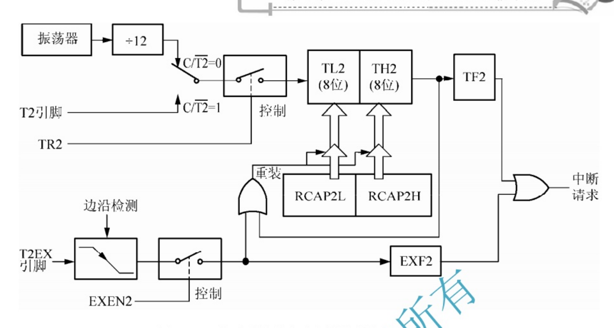


图 4-16　自动重装载方式逻辑结构框图(DCEN=0)

2) 捕获方式

捕获方式是指在一定条件下，自动将计数器 TH2 和 TL2 的数据读入 RCAP2H 和 RCAP2L，即 TH2 和 TL2 内容的捕获是通过捕获寄存器 RCAP2H 和 RCAP2L 来实现的。对于 89C52，其工作的逻辑结构框图如图 4-17 所示。

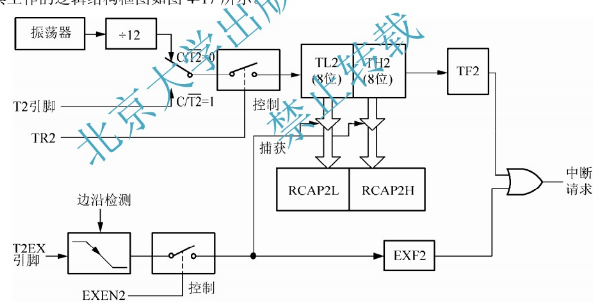


图 4-17　捕获方式逻辑结构框图

当 CP/$\overline{RL2}$=1 时，选择捕获方式。中断和捕获操作在下述两种情况下发生：

(1) 当定时器/计数器 T2 的寄存器(TH2、TL2)溢出时，溢出标志 TF2 置 1，申请中断。

(2) 当 EXEN2=1 且端口 T2EX(P1.1)出现下降沿信号时，打开捕获三态缓冲器，把(TH2、TL2)的内容自动读入到(RCAP2H、RCAP2L)中(捕获操作)。同时标志位 EXF2 置 1，申请中断。

若定时器/计数器 T2 的中断是被允许的，则无论发生 TF2=1 还是 EXF2=1，CPU 都会响应中断，此中断向量的地址为 2BH。响应中断后，应采用软件方式撤除中断申请，以正确地响应新的中断。TF2 和 EXF2 都是直接可寻址位，可采用指令“CLR TF2”和“CLR EXF2”撤除中断申请。

3) 波特率发生器方式

当 T2CON 中的 RCLK+TCLK=1 时，定时器/计数器 T2 将工作于波特率发生器方式，即其溢出脉冲用作串行口的时钟。定时器/计数器 T2 的波特率发生器方式下的逻辑结构框图如图 4-18 所示。在 T2CON 中，RCLK 选择串行通信接收波特率发生器，TCLK 选择串行通信发送波特率发生器。因此，发送和接收的波特率可以不同。

此时，定时器/计数器 T2 的输入时钟脉冲可由内部时钟电路决定，也可由外部时钟电路决定。

(1) 若 $C/\overline{T2}$=0，选用内部时钟，计数脉冲的频率为振荡器频率的 1/2。

(2) 若 $C/\overline{T2}$=1，选用外部时钟，该时钟由 T2(P1.0)端输入，每当外部信号产生下降沿时，计数器值加 1。外部脉冲频率不超过振荡器频率的 1/24。

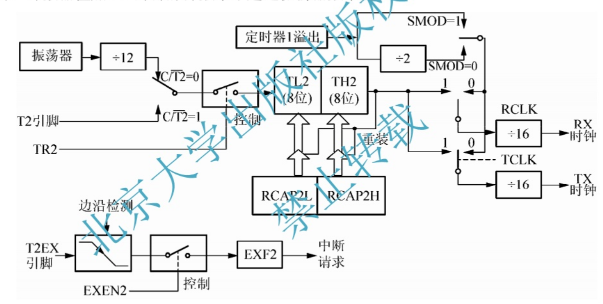


图 4-18 波特率发生器方式逻辑结构框图

由于脉冲溢出时，RCAP2H、RCAP2L 的内容会自动装载到 TH2、TL2 中，故波特率的值还决定于 RCAP2H、RCAP2L 的装载初值。

RCLK+TCLK 还用于选择定时器/计数器 T1 还是 T2 作串行通信的波特率发生器。由图 4-18 可看出，这两位的值用来控制两个电子开关的位置。值为 0 时，选用定时器/计数器 T1 作波特率发生器；值为 1 时，选用定时器/计数器 T2 作波特率发生器。

当定时器/计数器 T2 用作波特率发生器时，TH2 的溢出不会使 TF2 置位，即不会产生中断请求。因此，可以不禁止中断。

当定时器/计数器 T2 用作波特率发生器时，若 EXEN2 置 1，则端口 T2EX 的信号产生下降沿时，EXF2 将置 1，但不会发生重装载或捕获操作。这时，端口 T2EX 可以作为一个

额外的外部中断源。

在波特率发生器工作方式下，在 T2 计数过程中(即在 TR2=1 之后)，不能再读/写 TH2、TL2 的内容。如果读，则读出的结果不会精确(因为每个状态加 1)；如果写，则会影响 T2 的溢出而使波特率不稳定。在 T2 计数过程中，可以读出但不能改写 RCAP2H、RCAP2L 的内容。若需要访问 RCAP2H、RCAP2L，则应事先关闭定时器。

4.2.4　定时器/计数器的编程和使用

在定时器/计数器的 4 种工作方式中，方式 0 与方式 1 基本相同，只是计数器位数不同：方式 0 为 13 位计数器，方式 1 为 16 位计数器。方式 0 的存在是为了兼容 MCS-48，而且方式 0 的计数初值计算麻烦，所以在实际使用中，一般不使用方式 0，而使用方式 1。

例 4.2　设晶振频率 f_{osc}=6MHz，使用定时器/计数器 T1 以方式 1 产生周期为 2ms 的方波脉冲，并由 P1.0 输出，如图 4-19 所示。试以中断方式实现。

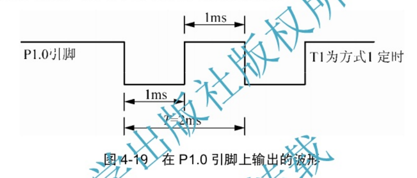


图 4-19　在 P1.0 引脚上输出的波形

解：方波的周期用定时器 T1 来确定，即在 T1 中设置一个初值，在初值的基础上进行计数，每隔 1ms 计数器 T1 溢出一次，即产生一次中断。CPU 响应中断后，在中断服务程序中对 P1.0 取反。

(1) TMOD 确定。TMOD 取值如图 4-20 所示。

D7	D6	D5	D4	D3	D2	D1	D0
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
0	0	0	1	×	×	×	×
T1				T0			

图 4-20　TMOD 取值

(2) 计算计数器的计数初值。要产生 2ms 的方波脉冲，只需在 P1.0 端以 1ms 为间隔，交替输出高低电平即可实现。为此，定时间应为 1ms。晶振频率 f_{osc}=6MHz，则一个机器周期为 2μs，设待求计数初值为 X，则

$$t=(2^L-X)\times t_C=(2^L-X)\times\frac{12}{f_{osc}}$$

其中：t为定时时间，L为计数器位数，X为计数初值，t_C为机器周期。

$1000\times10^{-6}=(2^{16}-X)\times2\times10^{-6}$

即 $500=2^{16}-X$

$X=2^{16}-500=10000\text{H}-1\text{F}4\text{H}$

$=0\text{FE}0\text{CH}=11111110\ 00001100\text{B}$

所以，初值为 TH1=0FEH，TL1=0CH。

(3) 采用中断方式。编程时打开全局和局部中断。设置 EA=1，ET1=1，以允许 T1 中断。

(4) 定时器的启动。由定时器控制寄存器 TCON 中的 TR1 位控制定时器的启动和停止。TR1=1，启动；TR1=0，停止。

(5) 程序设计。中断服务程序除了产生方波以外，还需将计数初值重新装入计数器，以便产生方波序列。

程序如下：



```
        ORG     0000H
        LJMP    MAIN                ; 主程序入口
        ORG     001BH
        LJMP    INTT1               ; T1 中断入口
        ORG     0030H
MAIN:
        MOV     SP, #60H            ; 修改堆栈指针
        MOV     TMOD, #10H          ; T1 为方式 1
        MOV     TH1, #0FEH          ; 设置计数初值
        MOV     TL1, #0CH
        SETB    EA                  ; 允许中断
        SETB    ET1                 ; 允许 T1 中断
        SETB    TR1                 ; 启动 T1
        SJMP    $                   ; 等待中断
INTT1:
        MOV     TH1, #0FEH          ; 重新设置初值
        MOV     TL1, #0CH
        CPL     P1.0                ; 定时 1ms 时间到, 输出取反
        RETI
        END
```


例 4.3 设晶振频率 f_{osc}=6MHz，使用定时器/计数器 T0 以方式 2 产生周期为 400μs 的方波脉冲，并由 P1.0 输出，如图 4-21 所示。试以中断方式实现。

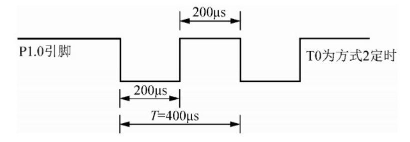


图 4-21 在 P1.0 引脚上输出的波形



解：方波的周期用定时器 T0 来确定，即在 T0 中设置一个初值，在初值的的基础上进行计数，每隔 200μs 计数器 T0 溢出一次，即产生一次中断。CPU 响应中断后，在中断服务程序中对 P1.0 取反。

(1) TMOD 确定。TMOD 取值如图 4-22 所示。

D7	D6	D5	D4	D3	D2	D1	D0
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
×	×	×	×	0	0	1	0
T1				T0			

图 4-22　TMOD 取值

(2) 计算计数器的计数初值。要产生 400μs 的方波脉冲，只需在 P1.0 端以 200μs 为间隔，交替输出高低电平即可实现。为此，定时间应为 200μs。晶振频率 f_{osc}=6MHz，则一个机器周期为 2μs，设待求计数初值为 X，则

$$200\times10^{-6} = (2^8-X)\times2\times10^{-6}$$

即

$$100 = 2^8-X$$

$$X = 2^8-100 = 100H-64H$$

$$= 9CH = 10011100B$$

所以，初值为 TH0=9CH，TL0=9CH。

(3) 采用中断方式。编程时打开全局和局部中断，设置 EA=1，ET0=1，以允许 T0 中断。

(4) 定时器的启动。由定时器控制寄存器 TCON 中的 TR0 位控制定时器的启动和停止。TR0=1，启动；TR0=0，停止。

(5) 程序设计。中断服务程序只需产生方波方波序列即可。

程序如下：

```
        ORG     0000H
        LJMP    MAIN            ；主程序入口
        ORG     000BH
        LJMP    INTT0           ；T1 中断入口
        ORG     0030H
MAIN:
        MOV     SP, #60H        ；修改堆栈指针
        MOV     TMOD, #02H      ；T0 为方式 2
        MOV     TH0, #9CH       ；设置计数初值
        MOV     TL0, #9CH
        SETB    EA              ；允许中断
        SETB    ET0             ；允许 T0 中断
        SETB    TR0             ；启动 T0
```

```
            SJMP    $               ; 等待中断
INTTO:
            CPL     P1.0            ; 定时 200μs 时间到，输出取反
            RETI
            END
```

例 4.4 门控制位 GATE 的应用——测量脉冲宽度。下面以 T1 为例，介绍门控制位 GATE 的应用。门控制位 GATE 可使定时器/计数器 T1 的启动计数受 INT1 的控制，当 GATE=1，TR1=1 时，只有 $\overline{INT1}$ 引脚输入高电平时，T1 才被允许计数，利用 GATE 位的这一功能(对于 T0 的 GATE 位也是一样，可使 T0 的启动计数受 $\overline{INT0}$ 的控制)，可测量 $\overline{INT1}$ 引脚(P3.3)上正脉冲的宽度(机器周期数)，其方法如图 4-23 所示。

P3.3 ($\overline{INT1}$)

对T1初始化 90H→TMOD GATE1=1

T1从0开始计数 测量脉冲宽度

读出T1的值 0→TR1，停止T1计数

图 4-23 利用 GATE 位测量正脉冲的宽度

参考程序如下：

```
            ORG     0000H
            LJMP    MAIN            ; 复位入口转主程序
            ORG     0030H
MAIN:       MOV     SP, #60H
            MOV     TMOD, #90H      ; 向 TMOD 写控制字，T1 为方式 1 定时，GATE=1
            MOV     TL1, #00H
            MOV     TH1, #00H
            JB      P3.3, $         ; 等待 INT1 降低
            SETB    TR1             ; 如果 INT1 为低，启动 T1，计数器开始计数
            JNB     P3.3, $         ; 等待 INT1 升高
            JB      P3.3, $         ; INT1 为高，等待 INT1 降低
            CLR     TR1             ; 停止 T1 计数
            MOV     A, TL1          ; T1 计数值送 A
            [将A中的T1计数值送到显示器显示]
            ⋮
```

执行以上程序，使 $\overline{INT1}$ 引脚上出现的正脉冲宽度以机器周期数的形式显示在显示器上。

4.3　中 断 系 统

中断系统在单片机系统中起着十分重要的作用。一个功能很强的中断系统，能大大提高单片机处理事件的能力，提高效率，增强实时性。80C51 系列微控制器(单片机)的中断功能较强，共设有 6 个中断源，6 个中断矢量：T0、T1、T2、$\overline{\text{INT0}}$、$\overline{\text{INT1}}$和一个串行通信中断矢量。有两级中断优先级，可实现两级中断嵌套。用户可以很方便地通过软件实现对中断的控制。

4.3.1　中断系统概述

1. 中断

程序执行过程中，允许外部或内部事件通过硬件中断程序的执行，使其转向处理外部或内部事件的中断服务程序中；完成中断服务程序后，CPU 继续原来被打断的程序，这样的过程称为中断。中断响应和处理过程如图 4-24 所示。

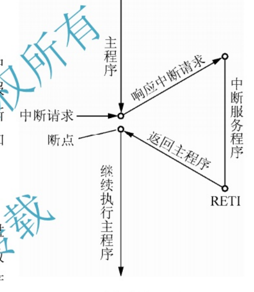


图 4-24　中断响应和处理过程

2. 中断源

能产生中断的外部和内部事件称为中断源，如键盘中断、A/D 转换器中断(当 A/D 转换器结束模拟量到数字量的转换而产生的中断)、定时中断(定时器/计数器产生的中断)、程序性中断(为调试程序而设置断点、单步工作等)。

对于每个中断源，不仅要求能发出中断请求信号，还要求这个信号能保持一定的时间，直至 CPU 响应这个中断请求后才能且必须撤销这个中断请求信号。这样既不会因 CPU 未及时响应而丢失中断申请信号，也不会出现多次重复中断的情况。

3. 中断优先级

几个中断源同时申请中断时，或者 CPU 正在处理某外部事件时，又有另一外部事件申请中断，CPU 必须区分哪个中断源更重要，从而确定优先处理哪个中断源，这就是中断优先级问题。就后者来说，优先级高的事件可以中断 CPU 正在处理的低级的中断服务程序，待完成了高级中断服务程序之后，再继续被中断的低级中断服务程序。

在 80C51 系列微控制器中，只有两级中断优先级。图 4-25 是 80C51 系列微控制器的中断系统结构示意图。

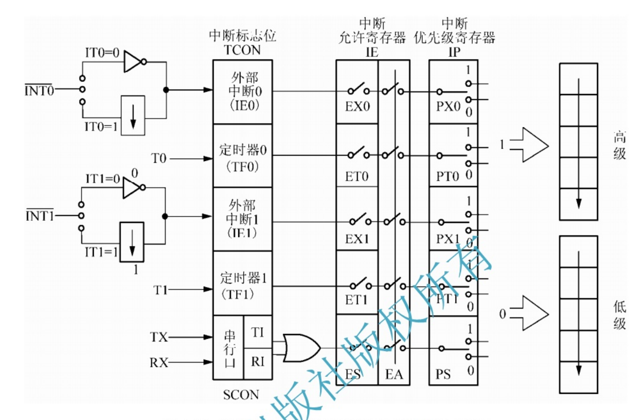


图 4-25　80C51 系列微控制器的中断系统结构示意图

4.3.2　中断的控制和操作

1. 80C52 系列微控制器中断源

80C51 系列微控制器中有 5 个中断源。80C52 系列微控制器中增加了一个定时器/计数器 T2 中断源，即有 6 个中断源。80C52 的 6 个中断源如下。

(1) $\overline{\text{INT0}}$ (P3.2)——外部中断 0。当 IT0(TCON.0)=0 时，低电平有效；当 ITO(TCON.0)=1 时，下降沿有效。

(2) $\overline{\text{INT1}}$ (P3.3)——外部中断 1。当 IT0(TCON.2)=0 时，低电平有效；当 IT1(TCON.2)=1 时，下降沿有效。

(3) TF0(P3.4)——定时器/计数器 T0 溢出中断。

(4) TF1(P3.5)——定时器/计数器 T1 溢出中断。

(5) RI，TI——串行中断。

(6) TF2、EXF2——定时器/计数器 T2 溢出中断。

微控制器各中断源提出的中断申请，如果得到微控制器的中断响应，则会自动转入各自固定的中断入口地址(中断矢量)见表 4-4。



表 4-4　80C51 系列微控制器的中断矢量表

中断源	中断标志	中断矢量	引脚	优先级
$\overline{\text{INT0}}$ 外部中断 0	IE0	0003H	P3.2	高
定时器/计数器 0 中断	TF0	000BH	P3.4	↓
$\overline{\text{INT1}}$ 外部中断 1	IE1	0013H	P3.3	↓
定时器/计数器 1 中断	TF1	001BH	P3.5	↓
串行中断	TI/RI	0023H	—	↓
定时器/计数器 2 中断	TF2/EXF2	002BH	P1.0/P1.1	低

2. 中断标志

$\overline{\text{INT0}}$、$\overline{\text{INT1}}$、T0 及 T1 的中断标志存放在定时器/计数器控制寄存器 TCON 中，串行口的中断标志存放在串行口控制寄存器 SCON 中，T2 的中断标志存放在定时器/计数器 T2 的控制寄存器 T2CON 中。

TCON 寄存器字节地址为 88H，其格式如图 4-26 所示。

位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
位功能	TF1	—	TF0	—	IE1	IT1	IE0	IT0

图 4-26　定时器/计数器控制寄存器 TCON 格式

SCON 寄存器字节地址为 98H，其格式如图 4-27 所示。

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位功能	—	—	—	—	—	—	TI	RI

图 4-27　串行口控制寄存器 SCON 格式

T2CON 寄存器字节地址为 0C8H, 其格式如图 4-28 所示。

位地址	CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H
位功能	TF2	EXF2	—	—	—	—	—	—

图 4-28　定时器/计数器控制寄存器 T2CON 格式

下面对中断标志及控制位的各位加以说明。

(1) TF2(T2CON.7)——定时器/计数器 T2 溢出标志。定时器/计数器 T2 溢出时置位，并申请中断。响应中断后需用软件清零。

(2) EXF2 (T2CON.6)——定时器/计数器 T2 外部标志。当 EXEN2=1，且 T2EX 引脚上出现下降沿而造成捕获或重装载时，EXF2 置位，并申请中断。响应中断后需用软件清零。

(3) TF1(TCON.7)——定时器/计数器 T1 溢出标志。硬件置位，响应中断后硬件清零。

不用中断时，需用软件清零。

(4) TF0(TCON.5)——定时器/计数器 T0 溢出标志。硬件置位，响应中断后硬件清零。不用中断时，需用软件清零。

(5) IE1(TCON.3)——外部中断请求 1 的中断标志。IE1=1 时，INT1 向 CPU 申请中断。

(6) IE0(TCON.1)——外部中断请求 0 的中断标志。IE0=1 时，INT0 向 CPU 申请中断。

(7) TI(SCON.1)——串行口发送中断标志。发送完一帧，硬件置位。响应中断后需用软件清零。

(8) RI(SCON.0)——串行口接收中断标志。接收完一帧，硬件置位。响应中断后需用软件清零。

(9) IT1(TCON.2)——外部中断请求触发方式控制位。

IT1=0，电平触发方式，引脚 $\overline{\text{INT1}}$ 上的外部中断请求输入信号为低电平有效，并使 IT1 置位。进入中断服务程序后，由硬件自动将 IE1 清零。

IT1=1，下降沿触发方式，引脚 $\overline{\text{INT1}}$ 上的外部中断请求输入信号电平从高到低的跳变有效，并使 IT1 置位。进入中断服务程序后，由硬件自动将 IE1 清零。

(10) IT0(TCON.0)——外部中断请求触发方式控制位。

IT0=0，电平触发方式，引脚 $\overline{\text{INT0}}$ 上的外部中断请求输入信号为低电平有效，并使 IT0 置位。进入中断服务程序后，由硬件自动将 IE0 清零。

IT0=1，下降沿触发方式，引脚 $\overline{\text{INT0}}$ 上的外部中断请求输入信号电平从高到低的跳变有效，并使 IT0 置位。进入中断服务程序后，由硬件自动将 IE0 清零。

3. 中断允许控制

中断允许和禁止由中断允许寄存器控制。中断允许寄存器 IE 是一个逐位定义的 8 位寄存器，既可字节寻址，也可位寻址。字节地址为 0A8H，位寻址的地址为 0A8H～0AFH。其格式如图 4-29 所示。

位地址	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
位功能	EA	—	ET2	ES	ET1	EX1	ET0	EX0

图 4-29 中断允许寄存器 IE 格式

中断允许寄存器 IE 中的各位为 0 时，禁止相应中断；为 1 时，允许相应中断。系统复位后，中断允许寄存器 IE 中各位均为 0，即此时禁止所有中断。下面对 IE 的各位加以说明。

(1) EX0(IE.0)——外部中断 0 中断允许位。

(2) ET0(IE.1)——定时器/计数器 T0 中断允许位。

(3) EX1(IE.2)——外部中断 1 中断允许位。

(4) ET1(IE.3)——定时器/计数器 T1 中断允许位。

(5) ES(IE.4)——串行口中断允许位。

(6) ET2(IE.5)——定时器/计数器 T2 中断允许位。

(7) EA(IE.7)——CPU 中断允许位(中断总允许位)。EA=1 时，开放所有中断；EA=0 时，屏蔽所有中断。



4. 中断优先级

在 80C51 系列单片机中有高、低两个中断优先级，通过中断优先级寄存器 IP 来设定。中断优先级寄存器 IP 是一个逐位定义的 8 位寄存器，既可字节寻址，也可位寻址。字节地址为 0B8H，位寻址的地址为 0B8H～0BFH。其格式如图 4-30 所示。

位地址	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H
位功能	—	—	PT2	PS	PT1	PX1	PT0	PX0

图 4-30　中断优先级寄存器 IP 格式

中断优先级寄存器 IP 中的各位为 0 时，相应中断为低优先级；为 1 时，相应中断为高优先级。系统复位后，IP 寄存器中各位均为 0，即此时全部中断设定为低优先级。

下面对 IP 的各位加以说明。

(1) PX0(IP.0)——外部中断 0 中断优先级控制位。

(2) PT0(IP.1)——定时器/计数器 T0 中断优先级控制位。

(3) PX1(IP.2)——外部中断 1 中断优先级控制位。

(4) PT1(IE.3)——定时器/计数器 T1 中断优先级控制位。

(5) PS(IP.4)——串行口中断优先级控制位。

(6) PT2(IP.5)——定时器/计数器 T2 中断优先级控制位。

在中断执行过程中，所有允许的高优先级中断可以中断低优先级中断的执行过程。

同级的中断不能相互中断。几个同级的中断源同时向 CPU 申请中断时，CPU 按硬件次序排定优先权，即依次为 $\overline{INT0}$、T0、$\overline{INT1}$、T1、串行口和 T2。

4.3.3　中断过程

1. 中断请求

中断请求就是中断源向 CPU 申请中断的过程，即建立中断请求标志位 IE0、IE1、TF0、TF1、TF2/EXF2、TI/RI 的过程。

1) 外部中断请求

外部中断源 $\overline{INT0}$、$\overline{INT1}$ 经由引脚 P3.2、P3.3 向 CPU 申请中断。外部中断请求有两种方式：电平触发方式和下降沿触发方式。通过设置触发方式控制位 IT0、IT1 进行选择。对于定时器/计数器 T2 的控制端 T2EX 来说，只有下降沿触发方式，控制端 T2EX 的下降沿经由引脚 P1.1 向 CPU 申请中断。

(1) 电平触发方式。若外部中断定义为电平触发方式，外部中断请求触发器的状态随着 CPU 在每个机器周期采样到的外部中断源输入线的电平变化而变化，这能提高 CPU 对外部中断请求的响应速度，但是也存在缺陷。这是因为在电平触发方式，单片机不锁存电平触发中断请求信号，会把每个机器周期采样到的外部中断源输入线上的逻辑电平直接赋值给中断标志寄存器。当中断请求被阻塞而没有得到及时响应时，将会被丢失。所以，要使电平触发的中断被 CPU 响应并执行，必须保证外部中断源输入线上的低电平维持到中断被执行为止。

当外部中断源被设定为电平触发方式时，在中断服务程序返回之前，外部中断请求输入必须无效(即外部中断请求为高电平)，否则，CPU 返回主程序后会再次响应中断。所以，在电平触发方式，需要 CPU 能够清除外部中断请求源。

(2) 下降沿触发方式。当 CPU 在一个机器周期中检测到中断源输入线上的电平为高电平，下一个机器周期检测到低电平，即电平信号有一个从高到低的跳变时，CPU 即置位中断标志，申请中断。所以，为确保检测到下降沿，中断源输入线上的高电平和低电平应至少各自保持一个机器周期。当中断标志寄存器锁存下降沿中断请求信号后，直到 CPU 响应中断并进入中断服务程序时，才由硬件自动清除。因此，在下降沿触发方式，当 CPU 正在执行同级或高级中断时，产生的外部中断同样会被记录到中断标志寄存器中。在同级或高级中断执行完毕后，该中断将被响应并执行。

外部中断请求标志存放在 IE0、IE1 中。

IE0 为 $\overline{\text{INT0}}$ 中断请求标志位，当 $\overline{\text{INT0}}$ 有中断请求时 IE0 置位，在 CPU 响应中断后，由硬件将 IE0 清零。

IE1 为 $\overline{\text{INT1}}$ 中断请求标志位，当 $\overline{\text{INT1}}$ 有中断请求时 IE1 置位，在 CPU 响应中断后，由硬件将 IE1 清零。

对于定时器/计数器 T2 的外部标志 EXF2 来说，当 T2EX 的下降沿来临，且 EXEN2 置位时，则置位 EXF2 并申请中断。在 CPU 响应中断后，只能由软件将 EXF2 清零。

2) 内部中断请求

当定时器/计数器 T0 计数产生溢出时，由硬件置位 TF0。当 CPU 响应中断后，再由硬件将 TF0 清零。

当定时器/计数器 T1 计数产生溢出时，由硬件置位 TF1。当 CPU 响应中断后，再由硬件将 TF1 清零。

当定时器/计数器 T2 计数产生溢出时，由硬件置位 TF2。当 CPU 响应中断后，只能由软件将 TF2 清零。

对于串行口中断来说，有两个标志位：TI、RI。当串行口发送完一个字节数据后，置位 TI；当串行口接收完一个字节数据后，置位 RI。CPU 响应中断后，只能由软件将 TI、RI 清零。

2. 中断响应

CPU 对中断请求进行判断，形成中断矢量，转入相应的中断服务程序的过程称为中断响应。只有满足规定要求的中断请求才能被 CPU 响应。

1) CPU 响应中断的基本条件

一个中断源的中断申请被响应，需满足以下基本条件。

(1) 有中断源提出中断申请。

(2) 中断总允许位 EA=1，即 CPU 开放中断。

(3) 申请中断的中断源的中断允许位为 1，即开放中断源。

(4) CPU 没有响应同级或更高优先级的中断。

(5) 当前指令执行结束。



(6) 如果正在执行的指令是 RETI 或是访问 IE、IP 指令，则 CPU 在执行 RETI 或访问 IE、IP 指令后，至少还要再执行一条其他指令后才会响应中断请求。

在接收中断申请时，如遇下列情况，硬件生成的长调用指令“LCALL”将被封锁：

(1) 正在执行同级或高优先级的中断服务程序。

(2) 所查询的机器周期不是执行当前指令的最后一个机器周期。

(3) 当前正在执行的指令是 RETI 或是访问 IE、IP 的指令。

2) 中断响应过程

(1) CPU 在每个机器周期的 S5P2 期间，顺序采样每一个中断源，建立中断请求标志。在下一机器周期按优先级的顺序查询各中断标志。若查询到某中断标志为 1，则按优先级的高低进行处理，即响应中断。

(2) 响应中断后，执行硬件生成的长调用指令“LCALL”，将程序计数器 PC 的内容压入堆栈保护，先低位地址，后高位地址，栈指针加 2。

(3) 将对应中断源的中断矢量地址装入程序计数器 PC，使程序转向该中断矢量地址，去执行中断服务程序。

由于中断矢量是固定的，两个中断矢量之间只有 8B 的存储空间，因此，通常在中断矢量指示的存储单元存放转移指令，由转移指令跳转到实际的中断服务程序去执行。

3. 中断处理

CPU 响应中断后，根据不同的中断源，形成不同的中断矢量，执行相应的中断服务程序。CPU 执行中断服务程序的过程，就是中断处理过程。中断处理一般包括：保护现场、中断服务和恢复现场三个部分，如图 4-31 所示。

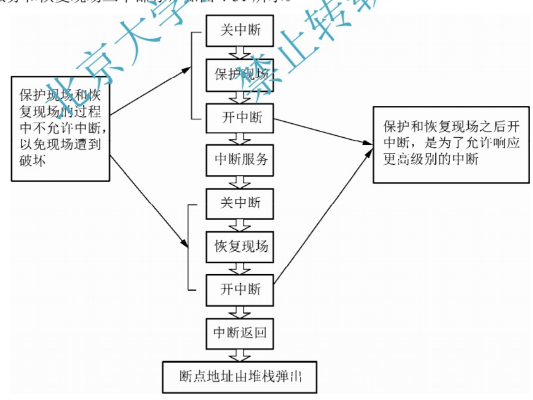


图 4-31　中断处理过程

中断服务程序类似于子程序，因此，首先应该是将该子程序用到的相关寄存器压入堆栈保护，以便中断返回时，主程序的现场能够恢复，一般用 PUSH 指令实现。中断服务是该中断要实现的操作或处理，是中断服务程序的主体。不同的中断源，有不同的中断需求，中断服务也就不一样。恢复现场程序段将先前压入堆栈的相关寄存器内容弹出，恢复主程序被中断的现场，以保证主程序的正常运行，一般用 POP 指令实现。在保护现场和恢复现场过程中，一般不允许被其他中断源中断，因此要关闭其他中断，否则容易引起中断的混乱。

4. 中断返回

1) 中断返回的过程

在中断服务程序的最后，必须安排一条中断返回指令 RETI。当 CPU 执行 RETI 指令时，自动完成下列操作。

(1) 将相应的优先级状态触发器清零。

(2) 恢复断点地址，即从堆栈中弹出栈顶的两个字节到程序计数器 PC，先弹出高位地址，后弹出低位地址，栈指针减 2，从而返回到断点处继续执行主程序；在中断返回后，自动完成开放同级中断和低级中断，以便允许同级和低级中断源申请中断。

2) 中断请求的撤除

CPU 响应中断请求，转向中断服务程序执行，在其执行中断返回指令(RETI)之前，中断请求信号必须撤除(复位)，否则将再一次引起中断而出错。

中断请求撤除的方式有三种。

(1) 由单片机内部硬件自动复位的：对于定时器/计数器 T0、T1 的溢出中断和采用下降沿触发方式的外部中断请求，在 CPU 响应中断后，由内部硬件自动复位中断标志 TF0 和 TF1、IE0 和 IE1，而自动撤除中断请求。

(2) 需用软件清除相应标志的：对于串行接收/发送中断请求和 80C52 中的定时器/计数器 T2 的溢出和捕获中断请求，在 CPU 响应中断后，内部无法硬件自动复位中断标志 TI 和 RI、TF2 和 EXF2，必须在中断服务程序中清除这些中断标志，才能撤除中断。

(3) 既无硬件也无软件撤除措施的：对于采用电平触发方式的外部中断请求，CPU 对 $\overline{\text{INT0}}$、$\overline{\text{INT1}}$ 引脚上的中断请求信号无法直接控制。因此，需要采取其他措施，即在引脚处加硬件电路来撤销中断申请。

3) 中断响应时间

中断响应时间是指从 CPU 检测到中断请求信号到转入中断服务程序所需要的时间。80C51 系列微控制器响应中断的最短时间为 2 个机器周期，最长为 8 个机器周期。

若 CPU 检测到中断请求信号时正好是一条指令的最后一个机器周期，且不是 RETI 指令或访问 IE、IP 指令，则不需要等待就可以立即响应，即由内部硬件执行一条长调用指令。若该指令需要 2 个机器周期，加上检测需要 1 个机器周期，一共需要 3 个机器周期就可以开始执行中断服务程序。



若中断检测时正在执行 RETI 指令或者访问 IE、IP 指令的第一个机器周期，这样包括检测在内需要 2 个机器周期(执行 RETI 指令或者访问 IE、IP 指令均需要 2 个机器周期)；若紧接着要执行的指令恰好是乘除法指令，其执行时间均为 4 个机器周期；再用 2 个机器周期执行一条长调用指令才能转入中断服务程序。这样，总共需要 8 个机器周期。

其他情况的中断响应时间都在 3～8 个机器周期之间。

4.3.4　外部中断源扩展

80C51 系列微控制器只提供了 2 个外部中断请求输入端 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ (定时器/计数器 T2 的外部输入 T2EX 在定时器/计数器 T2 用作波特率发生器时，可以作为一个外部中断源，且只有下降沿触发一种方式)。在实际应用中，如果需要使用多个外部中断源，就必须进行外部中断源的扩展。常用的几种外部中断源的扩展方法如下。

1. 定时器/计数器用于外部中断源的扩展

80C51 系列微控制器有 3 个定时器/计数器 T0、T1 和 T2，它们作为计数器使用时，计数输入端 T0(T1、T2)产生下降沿时将使计数器加 1。在计数方式下，如果把计数器预置为全 1，则只要在计数输入端(T0、T1 或 T2 输入端)出现一个下降沿脉冲就可以使计数器溢出，产生溢出中断。这时，就可以将计数输入端 T0(T1、T2)作为外部中断输入端。

例如，将定时器/计数器 T0 设置为工作方式 2，计数模式，计数初值为 0FFH，且允许中断。当计数输入端 T0 出现下降沿时，计数器的值加 1，产生溢出，从而申请中断，就相当于一个外部中断一样。

初始化程序如下：

```
ORG     0100H
MOV     TMOD,#06H        ；设置定时器/计数器 T0 为工作方式 2，计数模式
MOV     TH0,#0FFH        ；设置计数器初值
MOV     TL0,#0FFH
SETB    ET0              ；允许定时器中断
SETB    EA               ；开放 CPU 中断
SETB    TR0              ；启动定时器 T0
```

2. 查询方式用于外部中断源的扩展

当外部中断源较多时，可以采用查询方式扩展外部中断源。例如，把多个中断源通过 OC(或 OD)门线与后引入外部中断输入端($\overline{\text{INT0}}$、$\overline{\text{INT1}}$)，如图 4-32 所示。当任何一个中断源有中断申请时，其对应的 OC(或 OD)门输出为低，从而使 $\overline{\text{INT0}}$ 低电平有效，申请中断。在中断服务程序中用软件查询 P1 口的状态，便可以确定是哪一个中断源在申请中断，查询的次序由中断源的优先级次序决定。

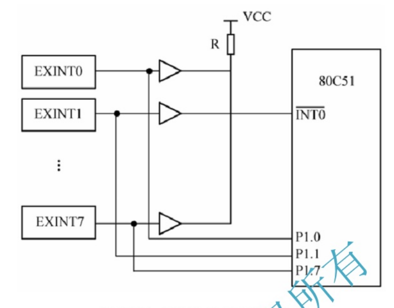


图 4-32　查询方式的中断源扩展

参考程序如下：

```
            ORG     0003H
            LJMP    INT0_EXT            ;跳转进入中断服务程序
            …       …
            ORG     0200H               ;中断服务程序入口地址
INT0_EXT:
            JNB     P1.0,IR0            ;是第 0 个中断请求，跳转进入中断服务程序 0
            JNB     P1.1,IR1            ;是第 1 个中断请求，跳转进入中断服务程序 1
            JNB     P1.2,IR2            ;是第 2 个中断请求，跳转进入中断服务程序 2
            JNB     P1.3,IR3            ;是第 3 个中断请求，跳转进入中断服务程序 3
            …       …
IR0:
            中断服务程序 0
IR1:
            中断服务程序 1
IR2:
            中断服务程序 2
IR3:
            中断服务程序 3
```

查询方式扩展外部中断源需要增加硬件，比较简单。但是当扩展的外部中断源数量较多时，查询时间较长，降低了中断的快速响应性能。

3. 用中断控制芯片扩展中断源

当需要扩展的外部中断源较多时，可以使用专用的中断控制器，如使用 8259 芯片来实现。一个 8259 芯片可以扩展 8 个中断源，经级联后，最多可以扩展 64 个中断源。



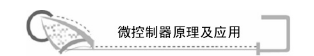
4.3.5　中断的编程和使用

例 4.5　一般出租车的计价器使用霍尔传感器 A44E 来检测车轮转动的圈数，通过计算得到行驶的里程数。每转动一圈，A44E 产生一个脉冲，将这一脉冲接到 MCU 引脚 P3.2($\overline{\text{INT0}}$)。使用这个脉冲的下降沿触发中断并计数。某出租车的轮胎型号为 195/65 R15，则轮胎周长为 3.14×(195×0.65×2+25.4×15)=1992.33≈2000(mm)。车轮转一圈的距离是 2m，则行驶里程为 2m×转动圈数。编写程序，用中断方式计算出租车行驶里程。数据存放在 R7、R6、R5 中，高位存放在 R7 中。

参考程序如下：

```
        ORG     0000H
        LJMP    MAIN            ; 主程序入口
        ORG     0003H
        LJMP    INTP0           ; 外部中断 0 入口
        ORG     0030H
MAIN:
        MOV     SP, #60H        ; 修改堆栈指针
        SETB    IT0             ; 设置下降沿触发方式
        SETB    PX0             ; 置外部中断 0 高优先级
        SETB    EX0             ; 允许外部中断 0
        SETB    EA              ; 开 CPU 中断
        MOV     R7, #0          ; 行车里程计数器清零
        MOV     R6, #0          ; 行车里程计数器清零
        MOV     R5, #0          ; 行车里程计数器清零
        SJMP    $               ; 等待中断

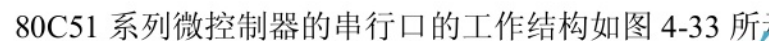
INTP0:
        PUSH    ACC             ; 保护现场
        PUSH    PSW
        MOV     A, R5           ; 读计数器低 8 位
        ADD     A, #2           ; 计数器低 8 位加 2
        MOV     R5, A           ; 重新保存计数结果
        CLR     A
        ADDC    A, R6           ; 计数器中 8 位加低 8 位的进位
        MOV     R6, A           ; 重新保存计数结果
        CLR     A
        ADDC    A, R7           ; 计数器高 8 位加中 8 位的进位
        MOV     R7, A           ; 重新保存计数结果
        POP     PSW             ; 恢复现场
        POP     ACC
        RETI
        END
```



4.4 串 行 接 口

在 80C51 系列微控制器中有一个串行接口(Serial Port)，一般简称串行口或串口，是一个全双工的异步串行通信接口，它可作 UART(Universal Asynchronous Receiver/Transmitter，通用异步接收和发送器)用，也可作同步移位寄存器用。所谓全双工，是指该接口可以同时进行接收和发送数据；所谓串行通信，是指数据一位接一位地顺序传送；所谓异步通信，是指接收和发送数据可以使用不同的波特率(时钟源)，以字符为数据传输单位，发送方传送字符的时间间隔不确定，双方遵循异步的通信协议。

4.4.1 串行口的结构


80C51 系列微控制器的串行口的工作结构如图 4-33 所示。

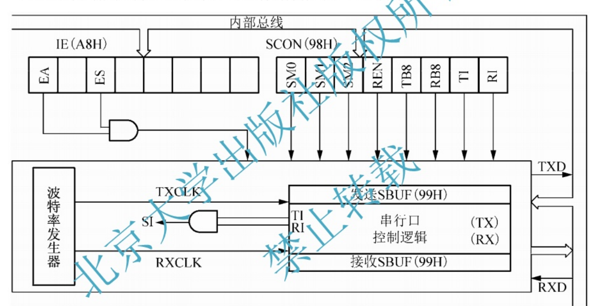


图 4-33 串行口的工作结构图

1. 波特率发生器

波特率发生器不是串行口结构的一部分，但却是串行口正常工作必不可少的。波特率发生器主要由定时器/计数器 T1、T2 及内部的一些控制开关和分频器所组成。波特率发生器向串行口送出的时钟信号为 TXCLK(发送时钟)和 RXCLK(接收时钟)，相应的控制波特率发生器的特殊功能寄存器有 TMOD、TCON、T2CON、PCON、TL1、TH1、TL2、TH2 等。

2. 串行口

串行口的内部组成如下。

(1) 接收寄存器 SBUF 和发送寄存器 SBUF。它们在物理上是隔离的，但是占用同一个地址 99H。可以通过访问特殊功能寄存器 SBUF 来访问接收缓冲器和发送缓冲器。接收缓冲器还具有双缓冲的功能，即它在接收第一个数据字节后，还能接收第二个数据字节。



但是，在它完成接收第二个数据字节之后，若第一个字节仍未取走，那么该字节数据将会丢失。

(2) 串行口控制逻辑。它接收来自波特率发生器的时钟信号——TXCLK(发送时钟)和RXCLK(接收时钟)；控制内部的输入移位寄存器将外部的串行数据转换为并行数据，控制内部的输出移位寄存器将内部的并行数据转换为串行数据输出。串行口控制逻辑还控制串行中断(RI 和 TI)。

(3) 串行口控制寄存器(SCON)。

(4) 串行数据输入/输出引脚。TXD(P3.1)为串行输入，RXD(P3.0)为串行输出。

4.4.2　串行口的特殊功能寄存器

串行口的特殊功能寄存器包括以下 3 个：串行口控制寄存器(SCON)、电源控制寄存器(PCON)和串行数据寄存器(SBUF)。

1. 串行口控制寄存器(SCON)

串行口控制寄存器 SCON 是一个逐位定义的 8 位寄存器，由它控制串行通信的方式选择、接收和发送，指示串行口的状态。寄存器 SCON 既可字节寻址也可位寻址，字节地址为 98H，位地址为 98H～9FH，其格式如图 4-34 所示。

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位功能	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

图 4-34　串行口控制寄存器 SCON 格式

下面对 SCON 的各位加以说明。

(1) SM0(SCON.7)、SM1(SCON.6)——串行口工作方式选择位，其对应的 4 种工作方式见表 4-5。

表 4-5　串行口的 4 种工作方式

SM0	SM1	工作方式	特点	波特率
0	0	方式 0	8 位移位寄存器	$f_{osc}/12$
0	1	方式 1	8 位 UART	可变
1	0	方式 2	9 位 UART	$f_{osc}/64$ 或 $f_{osc}/32$
1	1	方式 3	9 位 UART	可变

(2) SM2(SCON.5)——多机通信控制位。因为多机通信是在方式 2 和方式 3 下进行的，所以 SM2 位主要用于方式 2 或方式 3 中。当串行口以方式 2 或方式 3 接收时，如果 SM2=1，则只有当接收到的第 9 位数据(RB8)为 1 时，才将接收到的前 8 位数据送入 SBUF，并置位 RI，申请中断；当接收到的第 9 位数据(RB8)为 0 时，则丢弃接收到的前 8 位数据。当 SM2=0 时，则不论第 9 位数据是 1 还是 0，都将前 8 位数据送入 SBUF，并置位 RI，申请中断。

在方式 1 时，如果 SM2=1，则只有接收到有效的停止位，才能置位 RI。

在方式 0 时，SM2 只能为 0。

(3) REN(SCON.4)——允许串行接收位。由软件置位或清零。置位时，允许串行接收；清零时，禁止串行接收。

(4) TB8(SCON.3)——发送的第 9 位数据。在方式 2 和方式 3 时，TB8 是要发送的第 9 位数据，其值由软件置位或清零。在双机通信时，TB8 一般作为奇偶检验位使用；在多机串行通信中用来表示主机发送的是地址帧(TB8=1)还是数据帧(TB8=0)。

(5) RB8(SCON.2)——接收的第 9 位数据。在方式 2 和方式 3 时，RB8 存放接收到的第 9 位数据，可作为奇偶校验位或地址帧/数据帧的标志。在方式 1 时，如果 SM2=0，RB8 是接收到的停止位。在方式 0 时，不使用 RB8 位。

(6) TI(SCON.1)——发送中断标志位。在方式 0 时，当发送数据的第 8 位结束后，或在其他方式发送停止位后，由内部硬件置位 TI，表示一帧数据发送结束，并向 CPU 申请中断。CPU 在响应中断后，必须用软件方式将 TI 清零。如果不使用中断，则可用查询方式使用 TI，以判断一帧数据是否发送结束。

(7) RI(SCON.0)——接收中断标志位。在方式 0 时，当接收数据的第 8 位结束后，或在其他方式接收到停止位后，由内部硬件置位 TI，表示接收到完整的一帧数据，并向 CPU 申请中断。CPU 在响应中断后，必须用软件方式将 TI 清零零。如果不使用中断，则可用查询方式使用 RI，以判断是否接收到完整的一帧数据。

2. 电源控制寄存器(PCON)

电源控制寄存器 PCON 是一个逐位定义的 8 位寄存器，主要对微控制器的电源进行控制管理，目前仅有 5 位有定义，其中仅最高位 SMOD 与串行口控制有关。寄存器 PCON 的字节地址为 87H，不可位寻址，其格式如图 4-35 所示。

PCON	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SMOD	—	—	—	GF1	GF0	PD	IDL

图 4-35 电源控制寄存器 PCON 格式

SMOD 是串行通信波特率系数控制位。当 SMOD=1 时，使波特率加倍；当 SMOD=0 时，波特率不加倍。

3. 串行数据寄存器(SBUF)

串行数据(缓冲)寄存器 SBUF 包含在物理上隔离的两个 8 位寄存器，发送数据寄存器(发送缓冲寄存器)和接收数据寄存器(接收缓冲寄存器)，但是它们共用一个地址 99H，其格式如图 4-36 所示。

SBUF	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0

图 4-36 串行数据寄存器 SBUF 格式



写 SBUF(MOV A,SBUF)，访问发送数据寄存器；读 SBUF(MOV SBUF,A)，访问接收数据寄存器。

4.4.3　串行口的工作方式和多机通信方式

在控制寄存器中，SM0 和 SM1 位决定串行口的工作方式，SM2 位决定串行口应用于多机通信方式。

1. 方式 0

串行口的工作方式 0 为同步移位寄存器输入/输出方式。这种方式不能用于两个单片机之间的串行通信，常用于串行口外接串行输入/并行输出移位寄存器，以扩展并行 I/O 口。例如外接 74HC164，驱动数码管显示数据。这种方式下，数据传输波特率固定为 $f_{osc}/12$。数据由 RXD(P3.0)引脚输入/输出，同步移位时钟由 TXD(P3.1)引脚输出。接收/发送的是 8 位数据，传输时首先发送/接收最低位(LSB)，最后发送/接收最高位(MSB)。帧格式如图 4-37 所示。

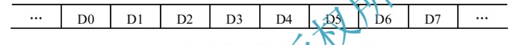


图 4-37　串行口工作方式 0 的帧格式

1) 方式 0 发送

当 CPU 执行一条将数据写入发送缓冲器 SBUF 的指令时，串行口开始把 SBUF 中的 8 位数据以 $f_{osc}/12$ 的固定波特率从 RXD 引脚串行输出，TXD 引脚输出同步移位脉冲，发送完 8 位数据后由硬件置位中断标志位 TI。若要再次发送数据，必须用指令将 TI 清 0。

2) 方式 0 接收

在方式 0 接收时，首先将接收中断标志位 RI 清 0。然后置位 REN，允许串口接收数据，CPU 即启动一次接收过程。在接收方式，RXD 为数据输入端，TXD 为移位脉冲信号输出端，接收电路以 $f_{osc}/12$ 的固定波特率采样 RXD 引脚的数据，当接收完 8 位数据后，中断标志置位 RI，表示一帧数据接收完毕，可以进行下一帧数据的接收。若要再次接收一帧数据，必须将上一帧数据取走，并使用指令将 RI 清 0。

复位时，SCON 被清 0，因此，默认的工作方式为方式 0。

2. 方式 1

工作方式 1 下的串行口为 8 位异步通信接口，传送一帧数据有 10 位，1 位起始位(低电平信号“0”)，8 位数据位(先传送最低位 LSB，后传送最高位 MSB)，一位停止位(高电平信号“1”)。这种方式下，数据传输波特率可变，由定时器/计数器 T1(或 T2)的溢出率和 SMOD(PCON.7)决定。接收数据由 RXD(P3.0)引脚输入，发送数据由 TXD(P3.1)引脚输出。帧格式如图 4-38 所示，S 表示起始位，P 表示停止位。

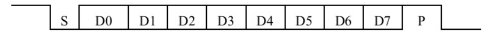


图 4-38　串行口工作方式 1 的帧格式

1) 方式 1 发送

当执行写 SBUF 指令时，在串行口由硬件自动加入起始位和停止位构成完整的 10 位

数据帧，在移位脉冲的作用下，通过 TXD 向外串行发送，当一帧数据发送完毕后，由硬件自动置位 TI。再次发送数据前，需用指令将 TI 清 0。

2) 方式 1 接收

在方式 1 接收时，首先将接收中断标志位 RI 清 0。然后置位 REN，允许串口接收数据；这时串行口不断采样 RXD 引脚，当采样到从 1 到 0 的跳变(下降沿)时，CPU 就认为接收到了起始位，开始了一次接收过程。随后在移位脉冲的控制下，数据从 RXD 引脚输入内部移位寄存器。

在方式 1 接收数据时，必须同时满足以下两个条件。

(1) RI=0。

(2) SM2=0 或接收到的停止位 P=1。

若以上两个条件中有一个不满足，则将不可恢复地丢失接收到的这一帧信息；若满足上述两个条件，则数据位装入 SBUF，停止位装入 RB8，且置位 RI。

接收这一帧之后，不论上述两个条件是否满足，即不论接收到的数据是否丢失，串行口都将继续检测 RXD 引脚上 1 到 0 的跳变，准备接收新的数据。

3. 方式 2

工作方式 2 下的串行口为 9 位异步通信接口，传送一帧数据有 11 位，1 位起始位(低电平信号“0”)，8 位数据位(先传送最低位 LSB，后传送最高位 MSB)，一位可编程位(由程序确定 0 或 1)，一位停止位(高电平信号“1”)。这种方式下，数据传输波特率只有两种：SMOD=0 时，波特率为 $f_{osc}/64$；SMOD=0 时，波特率为 $f_{osc}/32$。接收数据由 RXD(P3.0)引脚输入，发送数据由 TXD(P3.1)引脚输出。帧格式如图 4-39 所示，S 表示起始位，D8 表示 TB8/RB8，P 表示停止位。

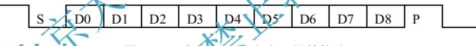


图 4-39 串行口工作方式 2 的帧格式

1) 方式 2 发送

当执行写 SBUF 指令时，在串行口由硬件自动加入起始位和停止位构成完整的 10 位数据帧，在移位脉冲的作用下，通过 TXD 向外串行发送，当一帧数据发送完毕后，由硬件自动置位 TI。再次发送数据前，需用指令将 TI 清 0。

2) 方式 2 接收

在方式 2 接收时，首先将接收中断标志位 RI 清 0。然后置位 REN，允许串口接收数据；这时串行口不断采样 RXD 引脚，当采样到从 1 到 0 的跳变(下降沿)时，CPU 就认为接收到了起始位，开始了一次接收过程。随后在移位脉冲的控制下，数据从 RXD 引脚输入内部移位寄存器。

在方式 2 接收数据时，必须同时满足以下两个条件：

(1) RI=0。

(2) SM2=0 或 SM2=1 且 RB8=1。

若以上两个条件中有一个不满足，则将不可恢复地丢失接收到的这一帧信息；若满足上述两个条件，则数据位装入 SBUF，第 9 位装入 RB8，且置位 RI。



接收这一帧之后，不论上述两个条件是否满足，即不论接收到的数据是否丢失，串行口都将继续检测 RXD 引脚上 1 到 0 的跳变，准备接收新的数据。

4. 方式 3

工作方式 3 下的串行口也是 9 位异步通信接口，传送一帧数据有 11 位，1 位起始位(低电平信号“0”)，8 位数据位(先传送最低位 LSB，后传送最高位 MSB)，一位可编程位(由程序确定 0 或 1)，一位停止位(高电平信号“1”)。可以看出，发送和接收的工作机制与方式 2 相同。但这种方式下，数据传输波特率是可变的，与方式 1 相同，也是由定时器/计数器 T1(或 T2)的溢出率和 SMOD(PCON.7)决定。接收数据由 RXD(P3.0)引脚输入，发送数据由 TXD(P3.1)引脚输出。帧格式如图 4-39 所示。

5. 多机通信

当微控制器工作在方式 2、3 时，具有多机通信功能，可以实现一台主机与多台从机的信息交流。需要注意的是，通信总是由主机发起，而从机之间是不能够直接通信的。图 4-40 为多机通信示意图，系统中左边的 80C51 为主机，其余的为 1～3 号从机，并保证每台从机在系统中的编号是唯一的。

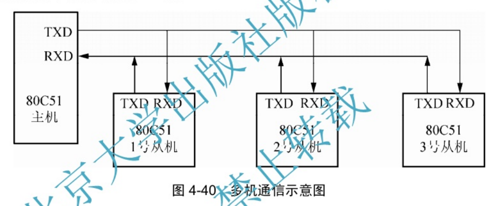


图 4-40　多机通信示意图

在主从式多机系统中，主机发出的信息帧有两类。

(1) 地址帧。主机发出的地址用来确定与其通信的从机，特征是串行发送的第 9 位数据 TB8=1，表示发出的一帧信息是地址帧。

(2) 数据帧。主机发送从机所需要的数据，特征是串行发送的第 9 位数据 TB8=0，表示发出的一帧信息是数据帧。

同样，对从机来说，接收的信息帧也有两类：

(1) 地址帧。当 SM2=1 时，从机接收到的一帧信息是地址帧。

(2) 数据帧。当 SM2=0 时，从机接收到的一帧信息是数据帧。

因此，对从机来说，在接收地址帧时，应使 SM2=1，以便接收主机发来的信息帧(TB8=1)，从而确定主机是否有意与自己通信。一旦确认接收到的是地址帧，则从机应立即使 SM2=0，以便接收主机发来的数据帧(TB8=0)。

单片机主从多机通信过程如下。

(1) 设置所有从机工作在方式 2 或方式 3，REN=1，SM2=1。

(2) 主机发出寻址信息(地址帧)，其中包括需要与其通信的 8 位从机地址，且设置第 9 位 TB8=1。

(3) 所有从机接收到地址帧后，使 RI=1。

(4) 各从机进行地址比较。如果接收到的地址与本机地址相同，则使 SM2=0，准备接收主机将要发来的数据帧；否则，保持 SM2=1，这样，从机就会对主机将要发来的数据帧不予理睬，直到接收到新的地址帧。

(5) 主机给已被寻址到的从机发送数据帧(包括控制指令和数据，且第 9 位 TB8=0)，实现主从机通信。

4.4.4 串行口的波特率发生器和波特率

波特率(Baud Rate)表示每秒传递的信息位的数量。波特率发生器用于控制串行口的数据传输速率。波特率的设定如下。

1. 方式 0 时串行口的波特率

方式 0 时的波特率由系统时钟频率 f_{osc} 所确定：

$$波特率=\frac{f_{osc}}{12}$$

2. 方式 2 时串行口的波特率

方式 2 时的波特率由系统时钟频率 f_{osc} 和 SMOD(PCON.7)所确定：

$$波特率=\frac{f_{osc}}{32}\times\frac{2^{SMOD}}{2}$$

当 SMOD=1 时，波特率=$\frac{f_{osc}}{32}$；当 SMOD=0 时，波特率=$\frac{f_{osc}}{64}$。

3. 方式 1 和方式 3 时串行口的波特率

方式 1 和方式 3 时的波特率由定时器/计数器 T1 或 T2 的溢出率和 SMOD(PCON.7)所确定。定时器/计数器 T1 和 T2 是可编程的，可选择的波特率范围比较大，因此，串行口的方式 1 和方式 3 是最常用的工作方式。

1) 定时器/计数器 T1($C/\overline{T}=0$)产生波特率

$$波特率=\frac{2^{SMOD}}{32}\times定时器/计数器T1的溢出率$$

定时器 T1 的溢出率与其工作方式有关：

(1) 定时器 T1 工作于方式 0(此时定时器 T1 相当于一个 13 位的计数器)：

$$溢出率=\frac{f_{osc}}{12}\times\frac{1}{(2^{13}-TC+X)}$$

式中：TC 为 13 位计数器初值；X 为中断服务程序的机器周期数，在中断服务程序中重新对定时器置数。

(2) 定时器 T1 工作于方式 1(此时定时器 T1 相当于一个 16 位的计数器)：

$$溢出率=\frac{f_{osc}}{12}\times\frac{1}{(2^{16}-TC+X)}$$

(3) 定时器 T1 工作于方式 2(此时定时器 T1 工作于一个 8 位可重装载的方式，用 TL1 计数，用 TH1 存储初值)：



$$溢出率=\frac{f_{osc}}{12}\times\frac{1}{(2^8-TH1)}$$

定时器 T1 的工作方式 2 是一种自动重装方式，不需要在中断服务程序中给 TL1 送数，由于没有中断引起的误差，所以应禁止定时器 T1 中断。这种方式是最常用的波特率设定方式。

2) 用定时器/计数器 T2 产生波特率

$$波特率=\frac{1}{16}\times定时器/计数器T2的溢出率$$

$$溢出率=\frac{f_{osc}}{2}\times\frac{1}{[2^{16}-(RCAP2H,RCAP2L)]}$$

式中：(RCAP2H,RCAP2L)为 16 位寄存器的初值(定时常数)。

4.4.5　串行口的编程和应用

使用微控制器的串行口，可以扩展微控制器的输入/输出端口，可以实现单片机之间的串行异步通信，也可以在多个单片机之间进行串行异步通信，还可以在单片机和 PC 之间进行串行异步通信。

1. *方式 0 时编程和应用*

例 4.6　使用串行口扩展数码管显示，电路图如图 4-41 所示。74LS164 在这里完成了串/并转换和驱动的双重任务。

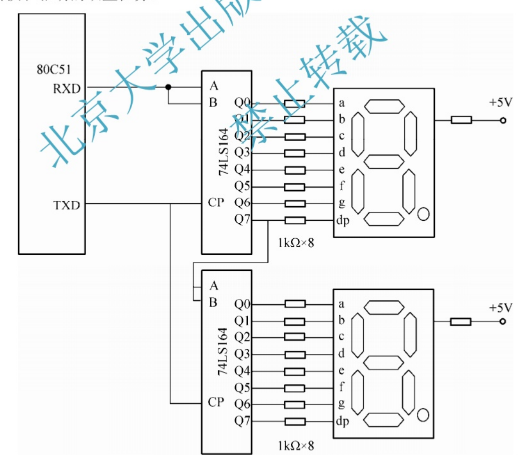


图 4-41　串行口扩展数码管电路图

参考程序如下：

```
X           EQU     9
Y           EQU     0

            ORG     0000H
            LJMP    MAIN                ;主程序入口
            ORG     0030H
MAIN:
            MOV     SP,#60H             ;修改堆栈指针
            MOV     SCON,#0             ;设置串行口工作在方式 0
            MOV     DPTR,#TAB           ;取显示的数字
            MOV     A,#X
            MOVC    A,@A+DPTR
            MOV     SBUF,A              ;将数字的编码送到串行口显示
            MOV     R0,#20              ;延时
            DJNZ    R0,$
            MOV     DPTR,#TAB           ;取显示的数字
            MOV     A,#Y
            MOVC    A,@A+DPTR
            MOV     SBUF,A              ;将数字的编码送到串行口显示
            jmp     $
TAB:
            DB      03H,9FH,25H,0DH,99H,49H,41H,1FH,01H,09H
            END
```


2. 方式 3 时编程和应用

例 4.7 设有如图 4-42 所示的甲、乙两台 MCU，以工作方式 2、全双工串行通信。每帧为 11 位，可程控的第 9 位数据位用于奇偶校验，波特率为 9600。

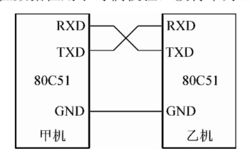


图 4-42 甲机、乙机串行通信电路图

编出能实现如下功能的程序。

甲机：每发送一帧信息，乙机对接收的数据进行奇偶校验，若校验正确，则乙机向甲机发出“数据发送正确”的信息(例中以 00H 作为回答信号)，甲机接收到该回答信号后再



发送下一字节；若奇偶校验错，则乙机发出“数据发送不正确”的信息(例中以 0AAH 作为回答信号)给甲机，要求甲机再次发送原数据，直至发送正确。甲机发送 16 个字节后就停止发送。

乙机：接收甲机发送来的数据并进行奇偶校验，与此同时发出相应的回答信息(即 00H 或 0AAH)，直到接收完 16 个字节为止。

解：能实现上述通信要求的甲、乙机的流程图如图 4-43、图 4-44 所示。

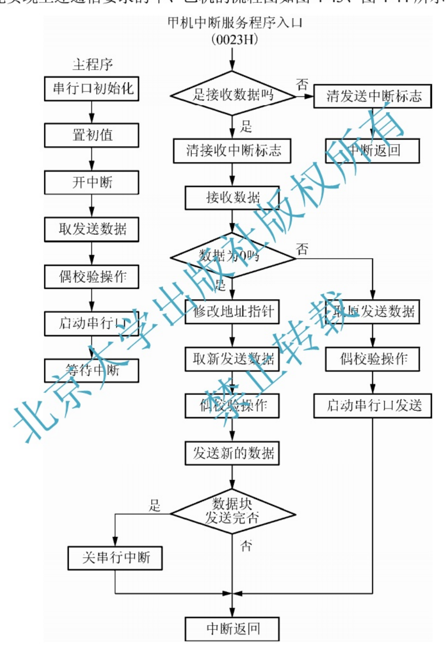


图 4-43　甲机发送接收流程图

图 4-44　乙机发送接收流程图

相应的程序如下。

(1) 甲机主程序。

```
ADDR1       EQU     40H             ; 数据块首地址
NUM         EQU     10H             ; 发送字节数
            ORG     0000H
            LJMP    MAIN            ; 转至主程序入口处
            ORG     0023H           ; 串行中断入口
            LJMP    INTSE1          ; 转至中断服务程序
```





```
MAIN:
        MOV     SP, #60H
        MOV     SCON, #0D0H         ; 置工作方式 3 并允许接收
        MOV     TMOD, #20H          ; 置定时器方式 2, 自动重装载
        MOV     TH1,  #0FDH         ; 波特率设置
        MOV     TL1,  #0FDH         ; 9600@11.0592MHz
        SETB    TR1                 ; 启动定时器
        SETB    EA                  ; CPU 开中断
        SETB    ES                  ; 允许串行口中断
        MOV     R1, #ADDR1          ; 置数据块指针
        MOV     R0, #NUM            ; 设置发送字节数初值
        MOV     A, @R1              ; 取第 1 个发送数据
        MOV     C, P                ; 奇偶标志位送 C
        MOV     TB8, C              ; 奇偶标志位送 TB8
        MOV     SBUF, A             ; 发送数据
        SJMP    $                   ; 等待中断
```

(2) 甲机中断服务程序。

```
INTSE1:
        JB      RI, LOOP1           ; 检测是否是接收中断, 是则转
        CLR     TI                  ; 是发送中断, 则先清除该标志
        SJMP    ENDT1               ; 转至 ENDT1 处
LOOP1:
        CLR     RI                  ; 是接收中断, 则先清除该标志
        MOV     A, SBUF             ; 取乙机的应答信息
        CJNE    A, #00H, LOOP2      ; 发送不正确转
        INC     R1                  ; 修改地址指针
        MOV     A, @R1              ; 取下 1 个发送数据
        MOV     C, P
        MOV     TB8, C              ; 将奇偶校验位送 TB8
        MOV     SBUF, A             ; 启动串行口, 发送新的数据
        DJNZ    R0, ENDT1           ; 数据发送完否, 未完返回
        CLR     ES                  ; 发送完毕, 禁止串行口中断
        SJMP    ENDT1               ; 转至中断返回处
LOOP2:
        MOV     A, @R1              ; 准备重发 1 次数据
        MOV     C, P
        MOV     TB8, C              ; 奇偶校验位送 TB8
        MOV     SBUF, A             ; 启动串行口, 重发 1 次数据
ENDT1:
        RETI                        ; 中断返回
        END
```

(3) 乙机主程序。

```
ADDR2       EQU     40H             ; 数据块首地址
NUM         EQU     10H             ; 发送字节数
            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    INTSE2
MAIN:
            MOV     SP, #60H
            MOV     SCON, #0D0H     ; 置工作方式 3, 允许接收
            MOV     TMOD, #20H      ; 置定时器方式 2, 自动重装载
            MOV     TH1,  #0FDH     ; 波特率设置
            MOV     TL1,  #0FDH     ; 9600@11.0592MHz
            SETB    TR1             ; 启动定时器
            SETB    EA              ; CPU 开中断
            SETB    ES              ; 允许串行口中断

            MOV     R1, #ADDR2      ; 置数据指针
            MOV     R0, #NUM        ; 传送 16 个字节
            SJMP    $               ; 等待中断
```

(4) 乙机中断服务程序。

```
INTSE2:
            JNB     RI, LOOP6       ; 不是接收中断则转
            CLR     RI              ; 是, 则清接收中断标志
            MOV     A, SBUF         ; 接收数据
            MOV     C, P            ; 判奇偶标志
            JC      LOOP4           ; 为奇数时转
            ORL     C, RB8          ; 为偶数时判 RB8
            JC      LOOP5           ; RB8 为 1 时(出错)转
LOOP3:
            MOV     @R1, A          ; 正确时则存入接收的数据
            MOV     A, #00H
            MOV     SBUF, A         ; 发送应答信息 00H
            INC     R1              ; 修改地址指针
            DJNZ    R0, ENDT2       ; 未完, 转返回
            CLR     ES              ; 已完, 则关串行口中断
ENDT2:
            RETI                    ; 中断返回
LOOP4:
            ANL     C, RB8          ; 为奇数时判 RB8
            JC      LOOP3           ; RB8 为 1 时(正确)转
```



```
LOOP5:
        MOV     A, #0AAH            ; 出错, 则发应答信息 0AAH
        MOV     SBUF, A
        SJMP    ENDT2               ; 转至中断返回处
LOOP6:
        CLR     TI                  ; 是发送中断, 则请 TI 标志
        SJMP    ENDT2               ; 转至中断返回处
        END
```



阅读材料

DP-51+单片机仿真实验仪

DP-51+单片机仿真实验仪(图 4-45)是一种功能强大的单片机应用技术学习、调试、开发工具。实验仪向用户提供了丰富的外围器件和设备接口，可使用户快速掌握单片机技术及其实用接口技术；同时，DP-51+单片机仿真实验仪具有长期保存用户应用程序的功能，可以作为实际应用系统设计的原型机，从而可以大大加快将理论设计投入实际应用的速度。



图 4-45　DP-51+单片机仿真实验仪

本 章 小 结

1. 并行 I/O 接口

80C51 系列微控制器有 4 个 8 位并行输入/输出端口，它们具有不同的功能，使用时要加以区分。一般来说，P0 口作为地址/数据分时复用的端口，可以输入/输出数据，或者通过外加的锁存器来输出地址。P2 口可以作为 16 位地址中的高 8 位地址输出。P3 口是一个双功能口，若不使用第二功能，可以作为一般的 I/O 口，其第二功能作为读写控制、中断信号以及串行口等。P1 口是通用的输入/输出口，由用户编程使用。

2. 定时器/计数器

80C51 系列微控制器的定时器/计数器是用来定时、计数的，是具有 2～3 个通道、4 种工作方式的可编程器件。

定时器/计数器的 3 个通道分别是 T0、T1 与定时器/计数器 T2，其中定时器/计数器 T2 仅 52 系列微控制器才有。

定时器/计数器内的核心器件是加 1 计算器，由两个特殊功能寄存器 TH 与 TL 组成。当定时器/计数器工作于定时方式时，加 1 脉冲由系统时钟 f_{osc} 经 12 分频后产生。当定时器/计数器工作于计数方式时，加 1 脉冲由 T0 或 T1 引脚直接提供。定时器/计数器工作于定时还是计数方式，取决于选择开关 $C/\overline{T}$，当 $C/\overline{T}=0$ 时工作于定时方式，$C/\overline{T}=1$ 时工作于计数方式。加 1 脉冲要经过启动开关 TR 才能到达加 1 计数器，当加 1 计数器溢出时，由硬件自动将中断标志 TF 置 1，以此向 CPU 发中断请求。

定时器/计数器的 4 种工作方式的主要区别在于加 1 计数器的位数，工作方式的选择由 TMOD 寄存器中的 M1M0 决定。

在使用定时器/计数器前必须进行初始化，即设置其工作方式。初始化一般进行如下工作：

(1) 设置工作方式，即设置 TMOD 中的 GATE、$C/\overline{T}$、M1M0 等各位。

(2) 计算加 1 计数器的计算初值 Count，并将计数初值 Count 送入 TH、TL 中。

计数方式：计数初值 Count=2^n-N。

定时方式：计数初值 Count=$2^n-T_d/T_{cy}$。式中，n=13、16、8、8 分别对应方式 0、1、2、3。

(3) 启动计数器工作，即将 TR 置 1，T0、T1 及 CPU 开中断。

定时器初始化后进行各种应用，如输出各种方波、实现实时时钟、对产品进行计数等。

3. 中断

由于中断源的请求，CPU 暂停当前程序而执行中断处理程序，完毕后返回原程序继续执行的过程称为中断。中断过程分为中断请求、响应、处理和返回 4 个阶段。

(1) 中断请求。中断请求是指中断源向 CPU 发中断信号。请求方法是：先由中断源将中断触发器置 1，然后由 CPU 在每个机器周期的最后去查询中断触发器，查询到 1，则转入中断响应阶段。否则继续执行下一条指令。80C51 系列微控制器的中断源有外部中断 $\overline{INT0}$ 和 $\overline{INT1}$、定时器中断 T0 和 T1、串行口中断。5 个中断源的中断触发器分别为 IE0、IE1、TF0、TF1 和 TI/RI。外部中断 $\overline{INT0}$ 和 $\overline{INT1}$ 使 IE0、IE1 置 1 的方法有两种：低电平与负跳变，两种中断请求方式可用 IT0、IT1 进行选择，IT0(或 IT1)=0 选择低电平中断方式，IT0(或 IT1)=1 选择负跳变中断方式。80C51 系列微控制器将中断触发器(IE0、IE1、TF0、TF1)、中断方式选择位(IT0、IT1)及定时器启动开关(TR0、TR1)，组合成定时器/中断控制寄存器 TCON。

(2) 中断响应。CPU 响应中断的条件是：CPU 执行完当前指令及允许中断。80C51 系



列微控制器允许中断是由中断允许寄存器 IE 中的各位决定的，各位取 1 允许中断，各位取 0 禁止中断。

(3) 中断处理。中断处理前应保护在主程序与中断处理程序中同时用到的寄存器或存储单元内容，称为保护现场，保护现场可用 PUSH 指令或换区的方法实现。现场保护后可执行中断处理程序，完成中断处理任务，最后应用 POP 指令或换区的方法恢复现场。

(4) 中断返回。中断返回是使用中断返回指令 RETI 实现的，RETI 指令的作用是将断点地址由堆栈弹回给 PC，使 CPU 返回到断点处执行源程序。

(5) 中断判优。当多个中断源同时发出中断请求时，CPU 先响应优先级最高的中断源，处理完毕后，再响应优先级次之的中断源，最后响应优先级最低的中断源，这就是中断判优的任务。80C51 系列微控制器有两个优先级：高优先级与低优先级，各中断源的优先级是通过优先级寄存器 IP 中各位实现的，IP 中的位取 1 设置高优先级，取 0 设置低优先级。

4. 串行接口

微控制器的串行通信有同步与异步通信。

微控制器串行通信的数据通路形式有单工、半双工、全双工 3 种方式。异步通信时可能会出现帧格式错、超时错等传输错误码，校验传输错误的方法有奇偶校验、和校验、循环冗余码校验、海明码校验。

与 80C51 系列微控制器串行通信有关的控制寄存器共有 3 个：SBUF、SCON 和 PCON。

80C51 系列微控制器的串行接口有 4 种通信方式。

方式 0 为同步通信方式，其波特率是固定的，为单片机晶振频率的 1/12，即 $BR=f_{osc}/12$。

方式 2 为异步通信方式，其波特率也是固定的，有两种。一种是晶振频率的 1/32，另一种是晶振频率的 1/64，即 $f_{osc}/32$ 和 $f_{osc}/64$。用公式表示为

$$BR=2^{SMOD}\times f_{osc}/64$$

方式 1 和方式 3 的波特率是可变的，其波特率由定时器 1 的计数溢出来决定，公式为

$$BR=2^{SMOD}\times T_d/32$$

设置定时器 T2 为波特率发生器工作方式，定时器 T2 的溢出脉冲经 16 分频后作为串行口发送脉冲、接收脉冲。发送脉冲、接收脉冲的频率称为波特率。其计算公式如下：

$$\text{波特率}=\frac{1}{16}\times\text{定时器 / 计数器T2的溢出率}$$

$$\text{溢出率}=\frac{f_{osc}}{2}\times\frac{1}{[2^{16}-(\text{RCAP 2 H,RCAP 2 L})]}$$

方式 1 是 10 位为一帧的异步串行通信方式。方式 2 和方式 3 是 11 位为一帧的异步串行通信方式，而第 9 位数据 D8 位既可作为奇偶校验位使用，也可作为控制位使用。在多机通信中经常把该位用作数据帧和地址帧的标志。SM2 为多级通信控制位，当 SM2=1 时，80C51 系列微控制器只接收第 9 个数据为 1 的地址帧，而对第 9 个数据为 0 的数据帧自动丢失；当 SM2=0 时，地址帧和数据帧全部接收。利用此特性可实现多机通信。

思考题与习题

1．80C51 系列 MCU 的 4 个 I/O 口在使用上有哪些分工和特点？何谓分时复用总线？P3 口的第二功能有哪些？

2．80C51 系列 MCU 端口 P0～P3 作通用 I/O 口时，在输入引脚数据时应注意什么？

3．为什么当 P2 口作为扩展程序存储器的高 8 位地址后就不能用作通用 I/O 口了？

4．定时器/计数器作定时用时，定时时间与哪些因素有关？作计数用时，对外部计数频率有何限制？

5．定时器/计数器 T0 工作在方式 3 时，由于 TR1 位已被 T0 占用，如何控制定时器/计数器 T1 的开启和关闭？

6．在 80C51 系列 MCU 系统中，已知时钟频率为 6MHz，选用定时器/计数器 T0 工作在方式 3，请编程实现 P1.0 和 P1.1 口分别输出周期为 1ms 和 400μs 的方波。

7．用 80C51 系列 MCU 的定时器测量某正单脉冲的宽度，采用何种方式可得到最大量程？若时钟频率为 6MHz，求允许测量的最大脉冲宽度是多少？

8．80C51 系列 MCU 的串口有几种工作方式？如何选择和设定？

9．什么是波特率？什么是溢出率？如何计算和设置串行通信的波特率？

10．为什么定时器/计数器 T1 用作串行口波特率发生器时，常采用方式 2？若已知系统时钟频率和通信波特率，应如何计算其初始值？

11．某异步通信接口，其帧格式组成为：1 个起始位 0、7 个数据位、1 个奇偶校验位和 1 个停止位 1。当该接口每分钟传输 1800 个字符时，计算其传输波特率。

12．80C51 系列 MCU 有几个中断源？各中断标志是如何产生的？又如何复位的？CPU 响应中断时，其中断入口地址各是多少？

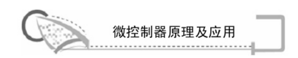
13．外部中断请求有哪两种触发方式？对下降沿触发信号和电平触发信号有什么要求？如何选择和设置？

第5章

微控制器的外部串行扩展技术

本章教学要点

知识要点	掌握程度	相关知识
I^2C 总线接口	掌握 I^2C 总线工作原理	I^2C 总线概述; I^2C 总线工作原理; I^2C 总线器件及工作模拟
SPI 总线接口	掌握 SPI 总线工作原理	SPI 总线概述; SPI 总线工作原理; SPI 总线器件及工作模拟





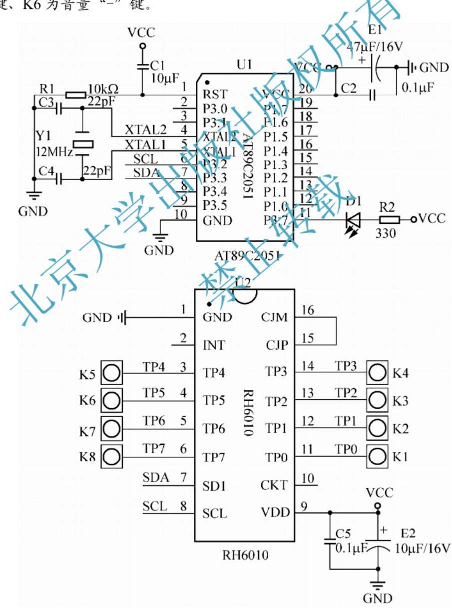
导入案例

RH6010 电视机触控面板应用

RH6010 是一款带 I^{2}C 总线接口的 8 通道电容式触摸感应控制开关，可替代传统机械开关。RH6010 可通过 I^{2}C 总线接口配置成多种模式，可广泛应用于灯光控制、玩具、家用电器等产品。RH6010 系列共有 2 个版本，其中 RH6010 面向低功耗；RH6010A 面向高灵敏度。

(1) 通过采用 AT89C2051 主控芯片与 RH6010 通信，配置和读取 RH6010 触摸状态，驱动 LED 动作。

(2) K1 为电源待机键，K2 为 MENU 键，K3 为频道“+”键、K4 为频道“－”键，K5 为音量“+”键、K6 为音量“−”键。



电视机触控面板参考电路图



近年来芯片间的串行数据传输技术被大量采用。由于数据的串行传输连接线少，采用串行总线技术可以使系统的硬件设计大大简化、系统的体积减小、可靠性提高。同时，系统的更改和扩充极为容易。

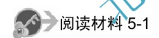
目前，MCU 应用系统中常用的串行扩展总线有：单总线(1-Wire Bus)、I^2C 总线(Inter IC Bus)、SPI 总线(Serial Peripheral Interface Bus)等。前两种总线是通过软件寻址来选通扩展器件，后一种则是通过并行 I/O 口线来选通扩展器件。

串行扩展总线的应用是 MCU 目前发展的一种趋势。80C51 系列 MCU 利用自身的通用并行线可以模拟多种串行总线时序信号，因此可以充分利用各种串行接口芯片资源(有些公司生产的 MCU 本身已经集成 I^2C、SPI 硬件接口，如 Silicon Labs 公司生产的 C8051F 系列 MCU)。

5.1　单总线接口

单总线接口是美国 DALLAS 公司(现已被 MAXIM 公司收购)的特有技术，它实现了在一条数据线上进行双向数据传输，最大限度地节省了通信线的数量，使系统布线更方便，成本更低，适合于多点分散系统。同时 DALLAS 公司推出了丰富的单总线产品，如数字温度传感器、信息纽扣(iButton)等。这种单总线芯片仅有一根信号线和一根地线。在信号线上综合了双向数字信号线和电源线的功能，每一块芯片都可提供一个独一无二的登记号，使用户可以灵活地构成不同功能的系统。而且，单总线产品基本为单芯片，成本低、可靠性高。例如 DS18B20，集传感器、温度转换、联网通信于一体。

由于单总线仅使用一根信号线进行双向数据传输，因此单总线的通信协议就比其他的串行通信协议要复杂很多，对通信时序和时间定时的要求非常严格。在一般的电子产品开发中往往要花费大量的精力和时间编写和调试单总线接口的底层及通信程序。



I^2C 总线的发展史

I^2C(Inter－Integrated Circuit)总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。I^2C 总线产生于 20 世纪 80 年代，最初为音频和视频设备开发，如今主要在服务器管理中使用，其中包括单个组件状态的通信。例如管理员可对各个组件进行查询，以管理系统的配置或掌握组件的功能状态，如电源和系统风扇。其可随时监控内存、硬盘、网络、系统温度等多个参数，增加了系统的安全性，方便管理。

I^2C 总线最主要的优点是其简单性和有效性。由于接口直接在组件之上，因此 I^2C 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本。I^2C 总线的另一个优点是，它支持多主控(multimastering)，其中任何能够进行发送和接收的设备都可以成为主总线。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。

5.2 I²C 总线接口

5.2.1 I²C 总线概述

I²C 总线是 PHILIPS 公司(现 NXP 公司)推出的一种高性能串行总线，具备以下特点：真正的多主机总线；无中央主机；总线自动仲裁；高低速器件同步功能。

I²C 总线只有两根双向信号线，一根是数据线 SDA，另一根是时钟线 SCL。所有连接到 I²C 总线上器件的数据线都接到 SDA 线上，各器件的时钟线均接到 SCL 线上。I²C 总线的基本结构如图 5-1 所示。

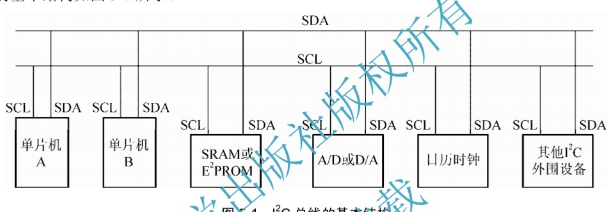


图 5-1 I²C 总线的基本结构

I²C 总线是一个多主机总线，即总线上可以有一个或多个主机，总线运行由主机控制。这里所说的主机是指启动数据的传送(发起始信号)、发出时钟信号、传送结束时发出终止信号的器件。通常，主机由各种 MCU 或其他微处理器充当。被主机寻址的器件叫从机，它可以是各种 MCU 或其他微处理器，也可以是其他器件，如存储器、LED 或 LCD 驱动器、A/D 或 D/A 转换器、时钟日历器件等。

I²C 总线的 SDA 和 SCL 是双向的，均通过上拉电阻接正电源，如图 5-2 所示。当总线空闲时，两根线均为高电平。连接到总线上的器件(相当于结点)的输出级必须是漏极或集电极开路的，任一器件输出的低电平都将使总线的信号变低，即各器件的 SDA 及 SCL 都是线“与”关系。SCL 线上的时钟信号对 SDA 线上各器件之间的数据传输起同步作用。SDA 线上数据的起始、终止及数据的有效性均要根据 SCL 线上的时钟信号来判断。

从 I²C 总线协议规范 2.0 版本开始，I²C 总线模式增加高速模式，现为 3 种工作模式。普通模式下，标准 I²C 总线系统的数据传输率最高为 100Kb/s；快速模式下数据传输率为 400Kb/s；高速模式下数据传输率可达 3.4Mb/s。连接的器件越多，总线的电容值越大，总线上允许的器件数以总线上的电容量不超过 400pF 为限。



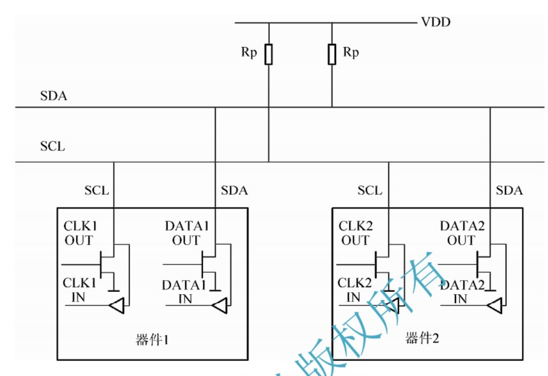


图 5-2　I²C 总线接口电路结构

每个接到 I²C 总线上的器件都有唯一的地址。主机与其他器件间的数据传送可以是由主机发送数据到其他器件，这时主机即为发送器。从总线上接收数据的器件则为接收器。

在多主机系统中，可能同时有几个主机企图启动总线传送数据。为了避免混乱，I²C 总线要通过总线仲裁，以决定由哪一台主机控制总线。不同主器件(欲发送数据的器件)分别发出的时钟信号在 SCL 线上“线与”产生系统时钟：其低电平时间为周期最长的主器件的低电平时间，高电平时间则是周期最短主器件的高电平时间。仲裁的方法是：各主器件在各自时钟的高电平期间送出各自要发送的数据到 SDA 线上，并在 SCL 的高电平期间检测 SDA 线上的数据是否与自己发出的数据相同。由于某个主器件发出的“1”会被其他主器件发出的“0”所屏蔽，检测回来的电平就与发出的不符，该主器件就应退出竞争，并切换为从器件。仲裁是在起始信号后的第一位开始，并逐位进行。由于 SDA 线上的数据在 SCL 为高电平期间总是与掌握控制权的主器件发出的数据相同，所以在整个仲裁过程中，SDA 线上的数据完全和最终取得总线控制权的主机发出的数据相同。在 80C51 系列 MCU 应用系统的串行总线扩展中，经常遇到的是以 80C51 系列 MCU 为主机，其他接口器件为从机的单主机情况。

5.2.2　I²C 总线工作原理

1. 数据位的有效性规定

I²C 总线进行数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定。只有在时钟线上的信号为低电平期间，数据线上的高电平或低电平状态才允许变化，如图 5-3 所示。

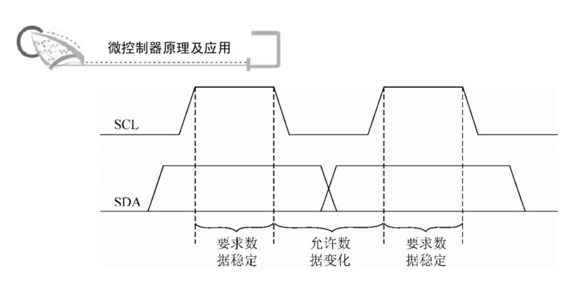


图 5-3　数据位的有效性规定

2. 起始和终止信号

根据 I^2C 总线协议的规定，SCL 线为高电平期间，SDA 线由高电平向低电平的变化表示起始信号；SDA 线由低电平向高电平的变化表示终止信号。起始信号和终止信号如图 5-4 所示。

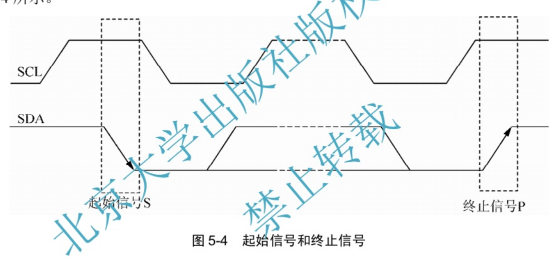


图 5-4　起始信号和终止信号

起始和终止信号都是由主机发出的，在起始信号产生后，总线就处于被占用的状态；在终止信号产生后，总线就处于空闲状态。

接收器件收到一个完整的数据字节后，有可能需要完成一些其他工作，如处理内部中断服务等，可能无法立刻接收下一个字节，这时接收器件可以将 SCL 线拉成低电平，从而使主机处于等待状态。直到接收器准备好接收下一个字节时，再释放 SCL 线使之为高电平，从而使数据传送可以继续进行。

3. 数据传送格式

利用 I^2C 总线进行数据传送时，传送的字节数是没有限制的，但是每一个字节必须保证是 8 位长度。数据传送时，先传送最高位(MSB)，每一个被传送的字节后面都必须跟随一位应答位(即一帧共有 9 位)，如图 5-5 所示。

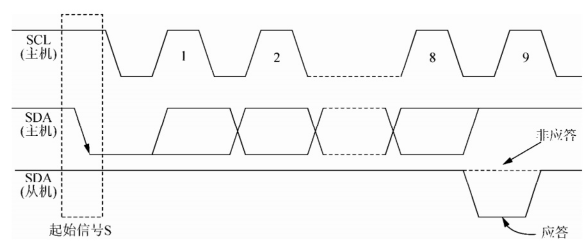


图 5-5　应答时序

由于某种原因从机不对主机寻址信号应答时(如从机正在进行实时性的处理工作而无法接收总线上的数据)，它必须将数据线置于高电平，而由主机产生一个终止信号以结束总线的数据传送。

如果从机对主机进行了应答，但在数据传送一段时间后无法继续接收更多的数据，从机可以通过对无法接收的第一个数据字节的“非应答”通知主机，主机则应发出终止信号以结束数据的继续传送。

当主机接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传送的信号。这个信号是由对从机的“非应答”来实现的。然后，从机释放 SDA 线，以允许主机产生终止信号。

4. 数据帧格式

I^2C 总线上传送的数据信号是广义的，既包括地址信号，又包括真正的数据信号。

I^2C 总线规定，在起始信号后必须传送一个从机的地址(7 位)，第 8 位是数据的传送方向位(读/写位，$R/\overline{W}$)，用“0”表示主机发送数据($\overline{W}$)，“1”表示主机接收数据(R)。每次数据传送总是由主机产生的终止信号结束。但是，若主机希望继续占用总线进行新的数据传送，则可以不产生终止信号，马上再次发出起始信号对另一从机进行寻址。因此，在总线的一次数据传送过程中，可以有以下几种组合方式。

(1) 主机向从机发送数据，数据传送方向在整个传送过程中不变，如图 5-6 所示。



S	从机地址	0	A	数据	A	数据	$A/\overline{A}$	P

图 5-6　主从发送格式

注：有阴影部分表示数据由主机向从机传送，无阴影部分则表示数据由从机向主机传送。A 表示应答，$\overline{A}$ 表示非应答(高电平)。S 表示起始信号，P 表示终止信号。

(2) 主机在发送完第一个字节后，立即读从机，如图 5-7 所示。

S	从机地址	1	A	数据	A	数据	$\overline{A}$	P

图 5-7　主从接收格式

(3) 在传送过程中，当需要改变传送方向时，起始信号和从机地址都被重复产生一次，但两次读/写($R/\overline{W}$)正好相反，如图 5-8 所示。

S	从机地址	0	A	数据	$A/\overline{A}$	S	从机地址	1	A	数据	$\overline{A}$	P

图 5-8　主从复合格式

由以上格式可见，无论哪种方式，起始信号、终止信号和地址均由主机发送，数据字节的传送方向则由寻址字节中 $R/\overline{W}$ 位规定，每个字节的传送都必须有应答。

5. I²C 总线的寻址

I²C 总线是多主机总线，总线上的各个主机都可以通过竞争获得总线，在竞争中获胜者将占有总线控制权。有权使用总线的主机如何对接收的从机寻址呢？I²C 总线协议有明确的规定：采用 7 位的寻址字节(寻址字节是起始信号后的第一个字节)。

1) 寻址字节的位定义

寻址字节的格式如图 5-9 所示。

MSB							LSB
从机地址							$R/\overline{W}$
D7	D6	D5	D4	D3	D2	D1	D0

图 5-9　I²C 寻址格式

D7～D1 位组成从机的地址。D0 位是数据传送方向位，为“0”时表示主机向从机写入数据，为“1”时表示主机由从机读数据。

主机发送地址时，总线上的每个从机都将这 7 位地址码与自己的地址进行比较，如果相同，则认为自己正被主机寻址，根据 $R/\overline{W}$ 位将自己确定为发送器或接收器。

从机的地址由固定部分和可编程部分组成。在一个系统中可能希望接入多个相同的从机，从机地址中可编程部分决定了该类器件可接入总线的最大数目。如果一个从机的 7 位地址有 4 位是固定位，3 位是可编程位，这时仅能寻址 8 个同样的器件，即可以有 8 个同样的器件接入该 I²C 总线系统中。

2) 寻址字节中的特殊地址

I²C 总线规定了一些特殊地址。其中两组固定地址编号 0000 和 1111 已被保留作为特殊用途，见表 5-1。

表 5-1　I²C 总线特殊地址表

地　址　位		$R/\overline{W}$	意　义
0　0　0　0	0　0　0	0	通用呼叫地址
0　0　0　0	0　0　0	1	起始字节
0　0　0　0	0　0　1	×	CBUS 地址
0　0　0　0	0　1　0	×	为不同总线的保留地址



续表

地　址　位		R/$\overline{W}$	意　义
0　0　0　0	0　1　1	×	保留
0　0　0　0	1　×　×	×	
1　1　1　1	1　×　×	×	
1　1　1　1	0　×　×	×	十位从机地址

起始信号后的第一字节的 8 位为“0000 0000”时，称为通用呼叫地址，即用于寻访接入 I^{2}C 总线上所有器件的地址。不需要从通用呼叫地址命令获取数据的器件可以不响应通用呼叫地址。否则，接收到这个地址后应作出应答响应，并把自己置为从机接收器方式以接收随后的各字节数据。另外，当遇到不能处理的数据字节时不应答，否则收到每个字节后都应作出应答响应。通用呼叫地址的用意在第二字节中加以说明，格式如图 5-10 所示。

第一字节(通用呼叫地址)									第二字节							LSB	
0	0	0	0	0	0	0	0	A	×	×	×	×	×	×	×	B	A

图 5-10　通用呼叫格式

当读/写位 B(R/$\overline{W}$)为“0”时，第二个字节的定义如下。

(1) 第二字节为 06H 时，所有能响应通用呼叫地址的从机器件复位，并由硬件装入从机地址的可编程部分。能响应命令的从机器件复位时不拉低 SDA 和 SCL 线，以免堵塞总线。

(2) 第二字节为 04H 时，所有能响应通用呼叫地址并通过硬件来定义其可编程地址的从机器件将锁定地址中的可编程位，但不进行复位。

如果第二字节的读/写位 B(R/$\overline{W}$)为“1”，则这两个字节命令称为硬件通用呼叫命令，也就是说这是由“硬件主器件”发出的。所谓硬件主器件，就是不能发送所要寻址从器件地址的发送器，如键盘等。制造这种器件时无法知道信息应向哪儿传送，所以它发出硬件呼叫命令，在第二字节的高 7 位说明自己的地址。接在总线上的智能器件，如 MCU 或其他微处理器能识别这个地址并与之传送数据。硬件主器件作为从机使用时，也用这个地址作为从机地址，格式如图 5-11 所示。

S	0000　0000	A	主机地址	1	A	数据	A	数据	A	P

图 5-11　硬件通用呼叫格式

在系统中另一种选择可能是系统复位时硬件主机器件工作在从机接收器方式，这时需要由系统中的主机先告诉硬件主机器件数据应送往的从机器件地址，当硬件主机器件要发送数据时，就可以直接向指定从机器件发送数据了。

3) 起始字节

起始字节是提供给没有 I^{2}C 总线接口的 MCU 查询 I^{2}C 总线时使用的特殊字节。

通常，MCU 可以通过两种方式接入 I^{2}C 总线。自身带有 I^{2}C 总线硬件接口的 MCU，

可以通过编程来响应由 I^2C 总线请求而产生的中断。对于不具备 I^2C 总线接口的 MCU，则必须通过软件不断地检测总线，以便及时响应总线的请求。MCU 检测总线或定时查询总线的次数越多，可用于执行其他控制功能的时间就越少。于是 MCU 的速度与硬件接口器件的速度就出现了较大的差别，为此，I^2C 总线上的数据传送要由一个较长的起始过程加以引导，如图 5-12 所示。

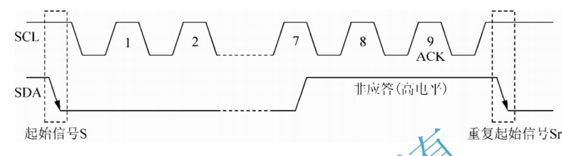
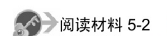


图 5-12 起始引导字节

引导过程由起始信号、起始字节、应答位、重复起始信号(Sr，a repeated START)组成。请求访问总线的主机发出起始信号后，发送起始字节(0000 0001)，另一个 MCU 可以用一个比较低的速率取样 SDA 线，直到检测到起始字节中的 7 个“0”中的一个为止。在检测到 SDA 线上的高电平后，MCU 就可以用较高的取样速率，以便寻找作为同步信号使用的第二个起始信号 Sr。

在起始信号后的应答时钟脉冲仅仅是为了和总线所使用的格式一致，并不要求器件在这个脉冲期间作出应答。


阅读材料 5-2

I^2C 总线的新进展

NXP 紧跟性能发展趋势，I^2C 速度由 100Kb/s 全面提升到 400Kb/s、1Mb/s 和 3.4Mb/s; NXP 开发出大量的集线器、中继器、多路复用器和开关等器件，将 I^2C 技术从芯片间二线通信的简单应用发展到功能强大而且全面的控制网络应用：扩展了总线容量，从 400pF 扩展到 4000pF，大大增加了总线可支持器件的数目。大量的总线管理器件能解决应用中的各种疑难问题；扩展了通信距离，不再仅仅是同一电路板上几个器件之间的通信，通信距离长达 1000m，可以通过线缆来通信；出色的热插拔功能，应用于系统内部及板机之间的通信，连接十分可靠、简洁。资料详尽的 I^2C 总线技术开发平台和越来越多的 I^2C 功能部件的成熟应用，使得 I^2C 软、硬件技术非常普及，可移植性、透明性的特点也全面提升到前所未有的水平。NXP 提供了丰富的 I^2C 总线管理器件，I^2C 功能器件及 I^2C 与 UART/SPI 之间的桥接芯片。I^2C 总线管理器件包括：I^2C 多路复用器和开关、I^2C 中继器、I^2C 集线器和扩展器、8 位并行转 I^2C 总线控制器、I^2C 电压电平变换器；I^2C 功能器件包括：实时时钟、LCD 驱动、I/O 扩展、AD 转换器、LED 调光、闪光器、数字温度传感器、数字 DIP 开关；I^2C 桥接芯片包括：I^2C/SPI 到 UART(含 IrDA 和 GPIO)、SPI 从机到 I^2C 主机(含 GPIO)、UART 到 I^2C 主机(含 GPIO)。



5.2.3　I^2C 总线器件介绍及工作模拟

随着微电子技术的发展，许多厂商不断推出 I^2C 总线接口器件，如 E^2PROM、A/D 转换器、D/A 转换器、LED 及 LCD 驱动器、日历时钟电路等。对于 80C51 系列 MCU，有一些品种在片上配置了 I^2C 总线接口(如 Silicon Labs 公司生产的 C8051F410)，但多数品种没有配置 I^2C 总线接口。这时可以利用通用并行 I/O 口线模拟 I^2C 总线接口的时序，使这些 MCU 可以与 I^2C 接口的器件通信。因此，在许多 MCU 应用系统中可以将 I^2C 总线的模拟传送技术作为常规的设计方法。

MCU 应用系统使用 I^2C 总线扩展时，大多数情况下为单主结构(只有一个 MCU 作为主器件的结构)的形式。在这种系统中，I^2C 总线只存在着单主机方式，总线数据的传送控制比较简单，没有总线的竞争与同步，只存在 MCU 对 I^2C 总线上各从器件的读(MCU 接收)、写(MCU 发送)操作。因此，主机可以采用不带 I^2C 总线接口的 MCU，如 STC89 系列的 MCU 等，利用软件实现 I^2C 总线的数据传送，即软件与硬件结合的信号模拟方式。

1. 典型信号模拟

为了保证数据传送的可靠性，标准 I^2C 总线的数据传送有严格的时序要求。I^2C 总线的起始信号、终止信号、发送“0”及发送“1”的模拟时序如图 5-13 所示。

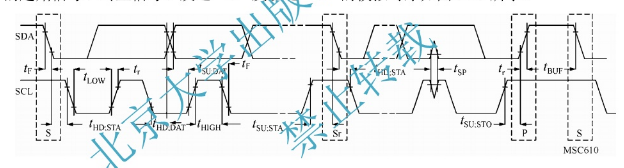


图 5-13　典型信号的时序模拟

表 5-2 所列为 I^2C 总线的时序特性。由表可见，除了 SDA 线、SCL 线的信号下降时间为最大值外，其他参数只有最小值。这表明在 I^2C 总线的数据传送中，可以利用时钟同步机制展宽低电平周期，迫使主器件处于等待状态，使传送速率降低。

表 5-2　I^2C 总线的时序特性表(普通模式时)

参数说明	符号	最小值	最大值	单位
新的起始信号前总线必需的空闲时间	t_{BUF}	4.7	—	μs
起始信号保持时间	$t_{HD;STA}$	4.0	—	μs
时钟的低电平时间	t_{LOW}	4.7	—	μs
时钟的高电平时间	t_{HIGH}	4.0	—	μs

续表

参数说明	符号	最小值	最大值	单位
起始信号建立时间(仅对重复起始信号)	$t_{SU;STA}$	4.7	—	μs
数据建立时间	$t_{SU;DAT}$	250	—	μs
SDA 线、SCL 线的信号下降时间	t_F	—	300	μs
终止信号建立时间	$t_{SU;STO}$	4.7	—	μs

对于一个新的起始信号，要求起始前总线的空闲时间 t_{BUF} 大于 4.7μs，而对于一个重复的起始信号，要求建立时间 $t_{SU;STA}$ 也大于 4.7μs。所以，图 5-13 中的起始信号适用于数据模拟传送中任何情况下的起始操作。起始信号到第一个时钟脉冲的时间间隔应大于 4.0μs。

对于终止信号，要保证有大于 4.7μs 的信号建立时间 $t_{SU;STO}$。终止信号结束时，要释放总线，使 SDA、SCL 维持在高电平上，在大于 4.7μs 后才可以进行第一次起始操作。在单主机系统中，为防止非正常传送，终止信号后 SCL 可以设置在低电平。

对于发送应答位、非应答位来说，与发送数据“0”和“1”的信号定时要求完全相同。只要满足在时钟高电平大于 4.0μs 期间，SDA 线上有确定的电平状态即可。

2. 典型信号模拟子程序

设主机采用 80C51 系列 MCU，晶振频率为 6MHz(即机器周期为 2μs)，则几个典型信号的模拟子程序如下。

1) 起始信号

```
STA:    SETB  P1.7
        SETB  P1.6
        NOP
        NOP
        CLR  P1.7
        NOP
        NOP
        CLR  P1.6
        RET
```

2) 终止信号

```
STP:    CLR  P1.7
        SETB  P1.6
        NOP
        NOP
        SETB  P1.7
        NOP
```



```
        NOP
        CLR P1.6
        RET
```

3) 发送应答位

```
ASK:    CLR P1.7
        SETB P1.6
        NOP
        NOP
        CLR P1.6
        SETB P1.7
        RET
```

4) 发送非应答位

```
NAS:    SETB P1.7
        SETB P1.6
        NOP
        NOP
        CLR P1.6
        CLR P1.7
        RET
```


阅读材料 5-3

串行总线与并行总线分析

串行总线相比于并行总线的主要优点是要求的线数较少。例如，用在汽车工业中的 LIN 串行总线只需要一根线来与从属器件进行通信，Dallas 公司的 1-Wire 总线只使用一根线来输送信号和电源。较少的线意味着所需要的控制器引脚较少。集成在一个微控制器中的并行总线一般需要 8 条或更多的线，线数的多少取决于设计中地址和数据的宽度，所以集成一个并行总线的芯片至少需要 8 个引脚来与外部器件接口，这增加了芯片的总体尺寸。相反，使用串行总线可以将同样的芯片集成在一个较小的封装中。

另外，在 PCB 板设计中并行总线需要更多的线来与其他外设接口，使 PCB 板面积更大、更复杂，从而增加了硬件成本。此外，工程师还可以很容易地将一个新器件加到一个串行网络中去，而且不会影响网络中的其他器件。例如，可以很容易地去掉总线上旧器件并用新的来替代。

串行总线的故障自诊断和调试也非常简单，可以很容易地跟踪网络中一个有故障的器件并用新器件替换而不会干扰网络。但另一方面，并行总线比串行总线速度快。

5.3 SPI 总线接口

串行外设接口(Serial Peripheral Interface，SPI)是由摩托罗拉公司(现飞思卡尔公司，Freescale Semiconductor)开发的全双工同步串行总线，该总线大量用在与 E²PROM、ADC、FRAM 和显示驱动器之类的慢速外设器件通信。

5.3.1 SPI 总线概述

SPI 实际上是一种串行总线接口标准，它可允许一次同步接收和发送 8 位数据，是一种全双工串行总线，其速度比 UART 串行接口要快。SPI 支持在同一总线上将多个从机连接到一个主机上。同一总线上也可以有多个主机，当两个或多个主机试图同时进行数据传输时，需要进行碰撞检测。

STC15F2K60S2 是 STC 公司生产的增强型的 80C51 系列 MCU，它集成了 SPI 接口，SPI 接口是一个全双工高速同步通信接口，既可以和其他微处理器通信，也可以与具有 SPI 兼容接口的器件，如存储器、A/D 转换器、D/A 转换器、LED 或 LCD 驱动器等进行同步通信。SPI 也可以在一个多主机系统中负责内部各个处理器之间的通信。SPI 接口有两种操作模式：主模式和从模式。在主模式中支持高达 3Mb/s 的速率(工作频率为 12MHz 时，如果 CPU 主频采用 20～36MHz，还可更高)，从模式时速度无法太快，速度在 $f_{osc}/8$ 以内较好。此外，SPI 接口还具有传输完成标志和写冲突标志保护功能。

STC15F2K60S2 的 SPI 接口功能方框图如图 5-14 所示。

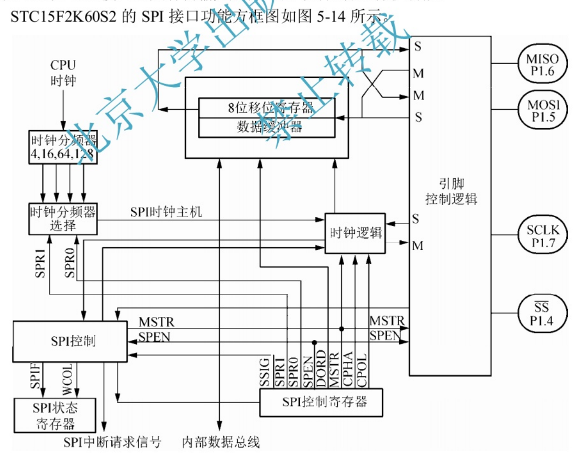


图 5-14 STC15F2K60S2 MCU 的 SPI 接口功能方框图



SPI 的核心是一个 8 位移位寄存器和数据缓冲器，数据可以同时发送和接收。在 SPI 数据的传输过程中，发送和接收的数据都存储在数据缓冲器中。

对于主模式，若要发送一个字节数据，只需将这个数据写到 SPIDAT 寄存器中。主模式下 $\overline{SS}$ 信号不是必需的。但是在从模式下，必须在 $\overline{SS}$ 信号变为有效并接收到合适的时钟信号后，方可进行数据的传输。在从模式下，如果一个字节传输完成后，$\overline{SS}$ 信号变为高电平，这个字节立即被硬件逻辑标志为接收完成，SPI 接口准备接收下一个数据。

任何 SPI 控制寄存器的改变将复位 SPI 接口，清除相关寄存器。

5.3.2　SPI 总线工作原理

1. SPI 接口的信号

SPI 接口由 MOSI(与 P1.3 共用)、MISO(与 P1.4 共用)、SCLK(与 P1.5 共用)和 $\overline{SS}$ (与 P1.2 共用)4 根信号线构成。SPI 接口的引脚可以通过外围设备切换控制寄存器 1 来改变。下面逐一介绍这 4 根信号线。

MOSI(Master Output Slave Input)：主机输出/从机输入数据线，用于主机到从机的串行数据传输。当器件的 SPI 设置为主机方式时，MOSI 是主机数据输出线；当器件的 SPI 设置为从机方式时，MOSI 是从机数据输入线。根据 SPI 规范，多个从机共享一根 MOSI 信号线。在时钟边界的前半周期，主机将数据发送到 MOSI 信号线上，从机在该边界处获取该数据。

MISO (Master Input Slave Output)：从机输出/主机输入数据线，用于实现从机到主机的数据传输。当器件的 SPI 设置为主机方式时，MISO 是主机数据输入线；当器件的 SPI 设置为从机方式时，MISO 是从机数据输出线。根据 SPI 规范，多个从机共享一根 MISO 信号线。当主机与一个从机通信时，其他从机应将其 MISO 引脚驱动置为高阻状态。

SCLK(SPI Clock)：串行时钟信号，对于主机来说，输出 SCLK；对于从机来说，输入 SCLK。SCLK 用于同步主机和从机之间在 MOSI 和 MISO 线上的串行数据传输。当主机启动一次数据传输时，自动产生 8 个 SCLK 时钟周期信号给从机。在 SCLK 的跳变处(上升沿或下降沿)移出一位数据。所以，一次数据传输可以传输一个字节的数据。

SCLK、MOSI 和 MISO 通常用于将两个或更多个 SPI 器件连接在一起。数据通过 MOSI 由主机传送到从机，通过 MISO 由从机传送到主机。SCLK 信号在主模式时为输出，在从模式时为输入。如果 SPI 接口被禁止，即特殊功能寄存器 SPCTL 中的 SPEN=0(复位值)，这些管脚都可作为 I/O 口使用。

$\overline{SS}$ (Slave Select)：从机选择信号，这是一个输入信号。主机用它来选择处于从模式的 SPI 模块。主模式和从模式下，$\overline{SS}$ 的使用方法不同。在主模式下，SPI 接口只能有一个主机，不存在主机选择问题。在该模式下 $\overline{SS}$ 不是必须的。主模式下通常将主机的 $\overline{SS}$ 引脚接 10kΩ的上拉电阻。每一个从机的 $\overline{SS}$ 接主机的 I/O 口，由主机控制电平高低，以便主机选择需要的从机进行通信。在从模式下，不论发送还是接收，$\overline{SS}$ 信号必须有效。因此在一次数据传输开始之前必须将 $\overline{SS}$ 设置为低电平。SPI 主机可以使用自己的 I/O 口来选择一个 SPI 器件作为当前的从机。

SPI 从机通过其$\overline{SS}$引脚确定是否被选择。如果满足下面的条件之一，$\overline{SS}$将被忽略：

(1) 如果 SPI 功能被禁止，即 SPEN(SPCTL.6)位为 0(复位值)。

(2) 如果 SPI 配置为主机，即 MSTR(SPCTL.4)位为 1，并且 P1.2/$\overline{SS}$配置为输出(P1M0.2=0、P1M1.2=1)。

(3) 如果$\overline{SS}$脚被忽略，即 SSIG(SPCTL.7)位为 1，该引脚配置为 I/O 口功能。

注意：即使 SPI 被配置为主机(MSTR=1)，仍然可以通过拉低$\overline{SS}$引脚配置为从机(如果 P1.2/$\overline{SS}$配置为输入且 SSIG=0)。要使能该特性，应当置位 SPIF(SPSTAT.7)。

2. SPI 接口的数据通信方式

STC15F2K60S2 的 SPI 接口的数据通信方式有 3 种：单主机-单从机方式、双器件方式(器件可互为主机和从机)和单主机-多从机方式。

1) 单主机-单从机方式

SPI 接口的单主机-单从机连接方式如图 5-15 所示。在图 5-15 中，从机的 SSIG (SPCTL.7)为 0，$\overline{SS}$用于选择从机。 SPI 主机可使用任何 I/O 口位(包括 P1.2/$\overline{SS}$) 来控制从机的$\overline{SS}$脚。主机 SPI 与从机 SPI 的 8 位移位寄存器连接成一个循环的 16 位移位寄存器。当主机程序向 SPDAT 写入一个字节时，立即启动一个连续的 8 位移位通信过程：主机的 SCLK 引脚向从机的 SCLK 引脚发出一串脉冲(8 个脉冲)，在这串脉冲的驱动下，主机 SPI 的 8 位移位寄存器中的数据移到了从机 SPI 的 8 位移位寄存器中。与此同时，从机 SPI 的 8 位移位寄存器中的数据移到了主机 SPI 的 8 位移位寄存器中。由此，主机既可向从机发送数据，又可从从机中读取数据。

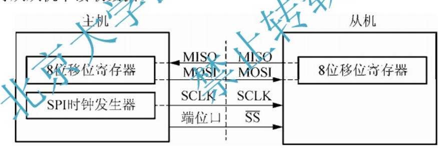


图 5-15　SPI 接口的单主机-单从机连接方式

2) 双器件方式

双器件方式也称为互为主从方式，其连接方式如图 5-16 所示。

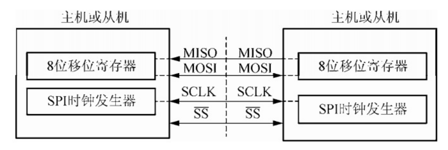


图 5-16　SPI 接口的双器件连接方式



在图 5-16 中，两个器件可互为主从。没有发生 SPI 操作时，两个器件都可配置为主机(MSTR=1)，将 SSIG 清 0 并将 P1.2/$\overline{SS}$配置为准双向模式。当其中一个器件启动传输时，可将 P1.2/$\overline{SS}$配置为输出并驱动为低电平，强制另一个器件变为从机。

双方初始化时都将自己设置成忽略$\overline{SS}$引脚的 SPI 从模式。当一方要主动发送数据时，先检测$\overline{SS}$引脚的电平，如果$\overline{SS}$引脚是高电平，就将自己设置成忽略$\overline{SS}$引脚的主模式。通信双方平时将 SPI 设置成没有被选中的从模式。在该模式下，MISO、MOSI、SCLK 均为输入，当多个 MCU 的 SPI 接口以此模式并联时不会发生总线冲突。这种特性在互为主从、一主多从等应用中很有用。注意，互为主从模式时，双方的 SPI 速率必须相同。如果使用外部晶体振荡器，双方的晶振频率也要相同。

3) 单主机-多从机方式

SPI 接口的单主机-多从机连接方式如图 5-17 所示。

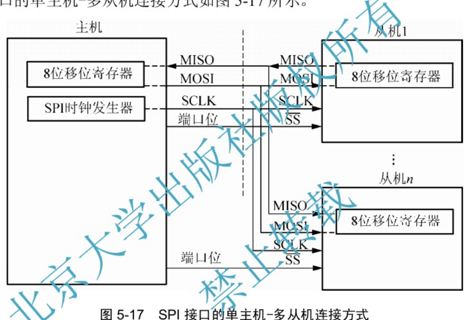


图 5-17　SPI 接口的单主机-多从机连接方式

在图 5-17 中，从机的 SSIG(SPCTL.7)为 0，从机通过对应的$\overline{SS}$信号被选中。SPI 主机可使用任何 I/O 口位(包括 P1.2/$\overline{SS}$)来控制从机的$\overline{SS}$。

STC15F2K60S2 进行 SPI 通信时，主机和从机的选择由 SPEN(SPCTL.6)、SSIG(SPCTL.7)、P1.2/$\overline{SS}$引脚和 MSTR(SPCTL.4)联合控制。主机和从机的模式选择见表 5-3。

表 5-3　SPI 主机/从机模式选择

SPEN	SSIG	$\overline{SS}$ P1.2	MSTR	主机模式或从机模式	MISO	MOSI	SCLK	备注
					P1.4	P1.3	P1.5	
0	×	P1.2	×	SPI 功能禁止	P1.4	P1.3	P1.5	SPI 禁止。P1.2/P1.3/P1.4/P1.5 作为普通 I/O 口使用
1	0	0	0	从机模式	输出	输入	输入	选择作为从机
1	0	1	0	从机模式未被选中	高阻	输入	输入	未被选中。MISO 为高阻状态，以避免总线冲突

续表

SPEN	SSIG	$\overline{SS}$ P1.2	MSTR	主机模式或从机模式	MISO P1.4	MOSI P1.3	SCLK P1.5	备注
1	0	0	1→0	从机模式	输出	输入	输入	P1.2/$\overline{SS}$配置为输入或准双向口。SSIG 为 0。如果选择$\overline{SS}$为低电平，则被选择为从机。当$\overline{SS}$变为低电平时，MSTR 将清 0 注：当$\overline{SS}$处于输入模式时，如被驱动为低电平且 SSIG=0，MSTR 位自动清 0
1	0	1	1	主(空闲)	输入	高阻	高阻	当主机空闲时 MOSI 和 SCLK 为高阻态以避免总线冲突。用户必须将 SCLK 上拉或下拉(根据特殊功能寄存器 SPCTL 的 CPOL 的取值)以避免 SCLK 出现悬浮状态
				主(激活)		输出	输出	作为主机激活时，MOSI 和 SCLK 为推挽输出
1	1	P1.2	0	从	输出	输入	输入	—
			1	主	输入	输出	输出	—

3. SPI 接口数据通信过程的注意事项

1) 器件 SPI 设置为从机模式

作为从机时，若 CPHA=0，SSIG 必须为 0，$\overline{SS}$引脚必须设置为低电平，并且在每个连续的串行字节发送完后重新设置为高电平。如果 SPDAT 寄存器在$\overline{SS}$有效(低电平)时执行写操作，那么将导致一个写冲突错误，WCOL 标志被置 1。CPHA=0 且 SSIG=0 时的操作未定义。

当 CPHA=1 时，SSIG 可以为 1 或 0。如果 SSIG=0，$\overline{SS}$引脚可在连续传输之间保持有效(即一直为低电平)。当系统中只有一个 SPI 主机和一个 SPI 从机时，这是首选配置。

2) 器件 SPI 设置为主机模式

在 SPI 通信中，数据传输总是由主机启动的。如果 SPI 使能(SPEN=1)且选择器件为主机，主机对 SPI 数据寄存器的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 引脚。

需要注意的是，主机可以通过将对应器件的$\overline{SS}$引脚驱动为低电平实现与之通信。写入主机 SPDAT 寄存器的数据从 MOSI 引脚移出发送到从机的 MOSI 引脚。同时，从机 SPDAT 寄存器的数据从 MISO 引脚移出发送到主机的 MISO 引脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志(SPIF)置位并产生一个中断(如果 SPI 中断使能)。主机和从机 CPU 的两个移位寄存器可以看作一个 16 位循环移位寄存器。



当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

4. 通过$\overline{SS}$改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式。$\overline{SS}$引脚可配置为输入或准双向模式。这种情况下，另外一个主机可将该引脚驱动为低电平，从而将该器件选择为SPI 从机并向其发送数据。

为了避免争夺总线，SPI 系统执行以下动作。

(1) MSTR 清零并且 CPU 变成从机。这样 SPI 就变成从机。MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式。

(2) SPSTAT 的 SPIF 标志位置位。如果 SPI 中断已被使能，则产生 SPI 中断。

用户程序必须一直对 MSTR 位进行检测，如果该位被一个从机选择清零而用户想继续将 SPI 作为主机，就必须重新置位 MSTR，否则将进入从机模式。

5. SPI 中断

如果允许 SPI 中断，发生 SPI 中断时 CPU 就会跳转到中断服务程序的入口地址 004BH 处执行中断服务程序。在中断服务程序中，必须把 SPI 中断请求标志清零。

6. 写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写入移位寄存器。当发送过程中对数据寄存器进行写操作时，WCOL 位将置位以指示数据冲突。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下一个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收到的数据将丢失。

WCOL 可通过软件向其写入 1 清零。

7. 数据格式

时钟相位控制位 CPHA 用于设置采样和改变数据的时钟边沿。时钟极性控制位 CPOL 用于设置时钟极性。不同的 CPHA，主机和从机对应的数据格式如图 5-18～图 5-21 所示。

SPI 接口的时钟信号线 SCLK 有 Idle 和 Active 两种状态：Idle 状态是指在不进行数据传输时(或数据传输完成后)SCLK 所处的状态：Active 是与 Idle 相对的一种状态。

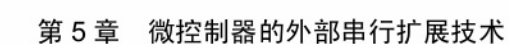
时钟相位位(CPHA)允许用户设置采样和改变数据的时钟边沿。时钟极性位 CPOL 允许用户设置时钟极性。

图 5-18　CPHA=0 时 SPI 从机传输格式

图 5-19　CPHA=1 时 SPI 从机传输格式

图 5-20　CPHA=0 时 SPI 主机传输格式





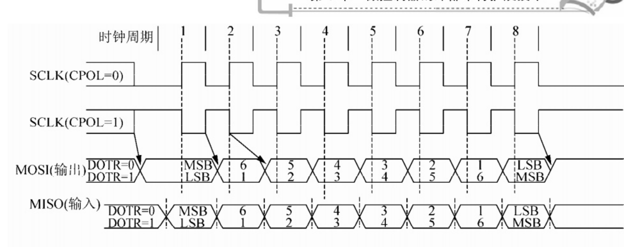


图 5-21　CPHA=1 时 SPI 主机传输格式

如果 CPOL=0，Idle 状态为低电平，Active 状态为高电平。

如果 CPOL=1，Idle 状态为高电平，Active 状态为低电平。

主机总是在 SCLK=Idle 状态时，将下一位要发送的数据置于数据线 MOSI 上。

从 Idle 状态到 Active 状态的转变，称为 SCLK 前沿。从 Active 状态到 Idle 状态的转变，称为 SCLK 后沿。一对 SCLK 前沿和后沿构成一个 SCLK 时钟周期，一个 SCLK 时钟周期传输一位数据。

8. SPI 时钟预分频器选择

SPI 时钟预分频器选择是通过 SPCTL 寄存器中的 SPR1-SPR0 位实现的。详见特殊功能寄存器 SPCTL 的介绍。


阅读材料 5-4

SPI 和 I^2C 总线比较

对于需要经常进行数据流传输的系统数据，SPI 是首选，因为它拥有较快的时钟速率。然而，对于系统管理活动，如读取温度传感器的读数和查询多个从器件的状态，或者需要多个主器件共存于同一系统总线上(系统冗余常会要求这一点)，或者面向低功耗应用，这时 I^2C 或 SMBus 将是首选接口。

1. SPI

SPI 是一种四线制串行总线接口，为主/从结构，四条导线分别为串行时钟(SCLK)、主出从入(MOSI)、主入从出(MISO)和从选($\overline{SS}$)信号。主器件为时钟提供者，可发起读从器件或写从器件操作。这时主器件将与一个从器件进行对话。当总线上存在多个从器件时，要发起一次传输，主器件将把该从器件选择线拉低，然后分别通过 MOSI 和 MISO 线启动数据发送或接收。

SPI 时钟速度很快，且没有系统开销。SPI 在系统管理方面的缺点是缺乏流控机制，无论主器件还是从器件均不对消息进行确认，主器件无法知道从器件是否繁忙。因此，必须设计软件机制来处理确认问题。同时，SPI 也没有多主器件协议，必须采用很复杂的软件和外部逻辑

来实现多主器件架构。每个从器件需要一个单独的从选择信号。总信号数最终为 $n+3$ 个，其中 n 是总线上从器件的数量。因此，导线的数量将随增加的从器件的数量按比例增长。同样，在 SPI 总线上添加新的从器件也不方便。对于额外添加的每个从器件，都需要一条新的从器件选择线或解码逻辑。

2. I^2C 总线

I^2C 是一种二线制串行总线接口，工作在主/从模式。主器件为时钟源。数据传输是双向的，其方向取决于读/写位的状态。每个从器件拥有一个唯一的 7 位或 10 位地址。主器件通过一个起始位发起一次传输，通过一个停止位终止一次传输。起始位之后为唯一的从器件地址，再后为读/写位。

I^2C 总线速度没有 SPI 那样快，但对于系统管理器件如温度传感器来说则非常理想。I^2C 存在系统开销，这些开销包括起始位/停止位、确认位和从地址位，但它因此拥有流控机制。主器件在完成接收来自从器件的数据时总是发送一个确认位，除非其准备终止传输。从器件在接收到来自主器件的命令或数据时总是发送一个确认位。当从器件未准备好时，它可以保持或延展时钟，直到其再次准备好响应。

I^2C 允许多个主器件工作在同一总线上。多个主器件可以轻松同步其时钟，因此所有主器件均采用同一时钟进行传输。多个主器件可以通过数据仲裁检测哪一个主器件正在使用总线，从而避免数据破坏。由于 I^2C 总线有两条导线，因此新从器件只需接入总线即可，而不需要附加逻辑。

5.3.3 SPI 总线器件介绍及工作模拟

1. SPI 相关的特殊功能寄存器

1) SPI 控制寄存器(SPCTL)

SPI 控制寄存器(地址为 0CEH，复位值为 00H)格式如图 5-22 所示。

位号	D7	D6	D5	D4	D3	D2	D1	D0
位名称	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

图 5-22 SPI 控制寄存器格式

(1) SSIG：$\overline{SS}$ 忽略控制位。

1：由 MSTR 位确定器件为主机还是从机。

0：由 $\overline{SS}$ 引脚用于确定器件为主机还是从机。$\overline{SS}$ 引脚可作为 I/O 口使用。

(2) SPEN：SPI 使能位。

1：SPI 使能。

0：SPI 被禁止，所有 SPI 管脚都作为 I/O 口使用。

(3) DORD：设定数据发送和接收的位顺序。

1：数据字的最低位(LSB)最先传送。

0：数据字的最高位(MSB)最先传送。



(4) MSTR：SPI 主/从模式选择位。具体选择方法参见表 5-3。

(5) CPOL：SPI 时钟极性。

1：SPI 空闲时 SCLK=1。SCLK 的前时钟沿为下降沿而后沿为上升沿。

0：SPI 空闲时 SCLK=0。SCLK 的前时钟沿为上升沿而后沿为下降沿。

(6) CPHA：SPI 时钟相位选择控制。

1：数据在 SCLK 的前时钟沿驱动到 SPI 口线，SPI 模块在后时钟沿采样。

0：数据在$\overline{SS}$为低(SSIG=0)时驱动到 SPI 口线，在 SCLK 的后时钟沿被改变，并在前时钟沿采样(注：SSIG=1 时的操作未定义)。

(7) SPR1：与 SPR0 联合构成 SPI 时钟速率选择控制位。

(8) SPR0：与 SPR1 联合构成 SPI 时钟速率选择控制位。SPI 时钟频率的选择见表 5-4。

表 5-4　SPI 时钟频率的选择

SPR1	SPR0	时钟(SCLK)	SPR1	SPR0	时钟(SCLK)
0	0	CPU_CLK/4	1	0	CPU_CLK/64
0	1	CPU_CLK/16	1	1	CPU_CLK/128

注：CPU_CLK 是 CPU 时钟。

2) SPI 状态寄存器(SPSTAT)

SPI 状态寄存器(地址为 0CDH，复位值为 00××××××B)格式如图 5-23 所示。

位号	D7	D6	D5	D4	D3	D2	D1	D0
位名称	SPIF	WCOL	—	—	—	—	—	—

图 5-23　SPI 状态寄存器格式

(1) SPIF：SPI 传输完成标志。当一次传输完成时，SPIF 被置位。此时，如果 SPI 中断被打开，即 ESPI(IE2.1)=1，EA(IE.7)=1，将产生中断。当 SPI 处于主模式且 SSIG=0 时，如果$\overline{SS}$为输入并被驱动为低电平，SPIF 也将置位，表示“模式改变”。SPIF 标志通过软件向其写入 1 而清零。

(2) WCOL：SPI 写冲突标志。当一个数据还在传输时，又向数据寄存器 SPDAT 写入数据，WCOL 将被置位。

WCOL 标志通过软件向其写入 1 而清零。

3) SPI 数据寄存器(SPDAT)

SPI 数据寄存器(地址为 0CFH，复位值为 00H)格式如图 5-24 所示。

位号	D7	D6	D5	D4	D3	D2	D1	D0
位名称	MSB	—	—	—	—	—	—	LSB

图 5-24　SPI 数据寄存器格式

D7～D0：保存 SPI 通信数据字节。其中，MSB 为最高位，LSB 为最低位。

2. SPI 总线器件介绍

TLC549 是美国德州仪器公司生产的 8 位串行 A/D 转换器芯片，通过 SPI 接口与 MCU 连接，从 CLK 输入的频率最高可达 1.1MHz。

TLC549 具有 4MHz 的片内系统时钟，片内具有采样保持电路，A/D 转换时间最长为 17μs，最高转换速率为 40000 次/s。

TLC549 的电源范围为+3～+6V，功耗小于 15mW，总失调误差最大为±0.5LSB，适用于电池供电的便携式仪表及低成本高性能的系统中。

1) 引脚功能

TLC549 有 8 个引脚，如图 5-25 所示。各引脚功能说明如下。

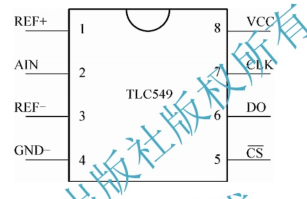


图 5-25 TLC549 引脚图

REF+：正基准电压输入端，2.5V≤REF+≤VCC+0.1V。

REF−：负基准电压输入端，−0.1V≤REF−≤2.5V，且要求 REF+－REF−≥1V。在要求不高时，也可将 REF−接地，REF+接 VCC。

AIN：模拟信号输入端，0≤AIN≤VCC，当 AIN≥REF+时，转换结果为全“1”(FFH)，AIN≤REF−时，转换结果为全“0”(00H)。

$\overline{CS}$：芯片选择输入端，低电平有效。

DO：数据串行输出端，输出时高位在前，低位在后。

CLK：外部时钟输入端，最高频率可达 1.1MHz。

2) TLC549 的工作模拟

TLC549 的时序如图 5-26 所示。$\overline{CS}$变为低电平时，TLC549 芯片被选中，同时从 DO 端输出前次转换结果的最高有效位 A7；接着自 CLK 端输入 8 个外部时钟信号，前 7 个 CLK 信号输出上次转换结果的 A6−A7 位。

在第 4 个 CLK 信号由高至低的跳变之后，片内采样/保持电路对输入模拟量采样开始，第 8 个 CLK 信号的下降沿使片内采样/保持电路进入保持状态并启动本次 A/D 开始转换。

TLC549 没有启动控制端，只要读走前一次数据后马上就进行新的转换，转换完成后就进入保持状态，转换时间为 36 个系统时钟周期，最大为 17μs。没有转换完成标志信号，只要采用延时操作即可控制每次读取数据的操作。



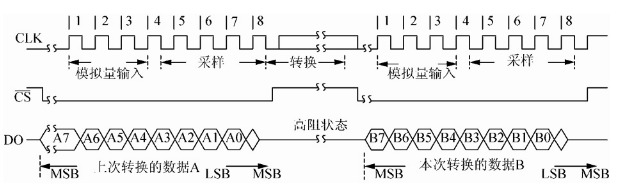


图 5-26　TLC549 的时序

3. TLC549 与 MCU 的连接

TLC549 与 MCU 的硬件连接如图 5-27 所示。采用 P1.0～P1.2 连接 TLC549 的串行接口。

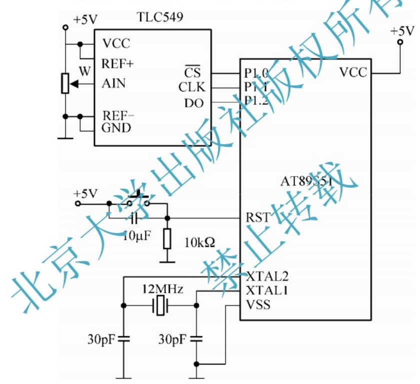


图 5-27　TLC549 与 MCU 的硬件连接

A/D 转换的汇编语言程序如下。

```
            DO    BIT  P1.2
            CLK   BIT  P1.1
            CS    BIT  P1.0
            …
TLC549_AD:  CLR  A              ; TLC549  A/D 转换子程序，转换结果在 A 中
            CLR  CLK
            MOV  R5, #08H
            CLR  CS             ; 选中 TLC549
```

```
LOOP:       SETB  CLK          ；产生时钟
            NOP
            NOP
            NOP
            NOP
            MOV  C, DO         ；读取 A/D 转换的一位数据
            RLC  A             ；左移进入 A
            CLR  CLK
            NOP
            NOP
            DJNZ  R5, LOOP     ；判断 8 次数据是否读完
            SETB  CS
            SETB  CLK
            RET
```

本 章 小 结

单总线接口将数据通信的引脚数目减少到最少，只需一个数据线便可以进行通信而不需要时钟同步线。使用单总线结构大大简化了电路设计，节约了引脚的使用，因此特别适合于 MCU 系统中。

I^2C 总线采用时钟(SCL)和数据(SDA)两根线进行数据传输，接口十分简单。串行 I^2C 接口 E^2PROM 电路具有体积小、接口简单、数据保存可靠、可在线改写和功耗低等特点，而且为低电压操作，并已经形成系列产品，在 MCU 系统中应用十分普遍。

SPI 总线是串行外围设备接口，是一种高速的、全双工、同步的通信总线，并且在芯片的管脚上只占用 4 根线。SPI 的通信原理很简单，它以主从方式工作，通常有一个主设备和一个或多个从设备，需要至少 4 根线。

思考题与习题

1. I^2C 总线的优点是什么？
2. I^2C 总线的起始信号和终止信号是如何定义的？
3. I^2C 总线的数据传送方向如何控制？
4. 常用的 I^2C 总线接口器件有哪些？
5. I^2C 总线的寻址方式如何？
6. I^2C 总线的数据传送时，应答是如何进行的？
7. 有哪些 MCU 具备 I^2C 总线接口？
8. 简述 STC15F2K60S2 的 SPI 接口的特点。

第 6 章

微控制器的典型外围接口技术

本章教学要点

知识要点	掌握程度	相关知识
键盘接口技术	了解键盘的工作原理; 掌握键盘的工作方式及接口电路	独立式键盘及其接口; 矩阵式键盘及其接口
显示器接口技术	理解 LED 的结构和显示原理; 掌握显示器接口电路及应用编程	静态显示控制方式; 动态显示控制方式; 动态显示的接口电路与编程实现
DAC 与 ADC 接口	掌握微控制器应用系统中的数/模和模/数转换接口电路设计的方法	D/A 转换电路的设计与应用; 串行 D/A 转换器及应用简介; A/D 转换器及其应用简介



导入案例

键盘的发展历史

1971 年年底的一天，英国工程师雷·汤姆林森(Ray Tomlinson)在互联网的前身 ARPAnet 系统上编写了一个程序，经过几次尝试后，程序成功运行了，一段信息呈现在了另一台电脑的屏幕上。汤姆林森当时还没有意识到，这是世界上第一封真正意义上的电子邮件。20 多年后，当电子邮件成为日常生活不可缺少的一部分时，作为历史上伟大瞬间缔造者的汤姆林森被问及邮件的内容时，他答道“或许是 QWERTYUIOP”。与人们想象的“Hello”，“How are you”一类内容不同，汤姆林森当时只把这封邮件当成一次普通的程序运行尝试，随手输入了计算机键盘上第一行的 10 个字母。很多人第一次接触电脑键盘时，都会问这样的问题，电脑上键盘的第一行为什么是 QWERTYUIOP，而不是按照字母顺序排列的 ABCDEFGHIJ?如果那样的话就不需要花时间记住每一个字母键的位置了。这个问题要由键盘最早的发明者来回答。你或许不知道，键盘的历史比计算机的历史还要早很多，虽然世界上第一台计算机在 20 世纪 40 年代出现，个人使用的小型计算机到 20 世纪 70 年代才开始出现，但是作为现在计算机一个组件的键盘在 19 世纪 70 年代就出现了。键盘最初是出现在 1868 年美国人 Christopher Latham Sholes 发明的机械打字机上，它是世界上第一台商用的机械打字机，使人们彻底告别了“活字印刷”的时代，当时“这台神奇的机器可以将一个个字母整齐、准确地打在羊皮纸上”，马上吸引了众多工厂购买专利进行生产，大量政府、公司职员去购买使用。



机械打字机

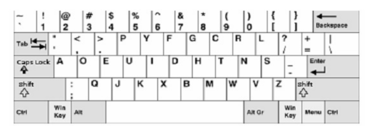
最初这台打字机的键盘分布就是按照 ABCDEFGHIJ 的顺序，但是在实际中却出现了问题。你或许会猜是不是这种排布不利于提高打字速度？正相反，这种排布的打字速度太快了！受当时机械设备的限制，如果打字员打字的速度过快，打字机相邻键杆就会撞在一起而发生卡壳。所以 Christopher Latham Sholes 对他的发明进行了改进，人为地降低了一些常用字母的输入速度，设计了 QWERT 式的键盘，也就是我们现在使用的键盘。



~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Delete
Tab Q W E R T Y U I O P { [}] | \
Caps A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Alt Alt Ctrl

QWERTY 键盘

后来打字机的设计水平得到了提高，卡壳的现象几乎不再出现，到了 20 世纪中期，电子键盘代替了机械键盘，对输入速度过快的担心完全成了杞人忧天，QWERT 式的键盘几乎成了明日黄花。与此同时，经过了蒸汽机时代和爱迪生时代后，各种各样的机器和新发明走到了人们的身边，同样的机器，怎样设计可以使用户使用更加方便，怎样设计可以使这些产品更加人性化逐渐成了一个棘手的问题，于是出现了一门新的学问——人机工程学。从人机工程学角度分析，QWERT 式的键盘或许是人类历史上最糟糕的发明之一，有很多缺陷。首先，英文 26 个字母在实际中使用的频率是不同的，最常见的字母 e 出现的频率高达 12.702%，字母 t 也有 9.056%。与之相比，字母 q 出现的频率仅有 0.095%，最少出现的 z 则只有 0.074%。按照键盘打字的指法，在键盘的三行中，中间一行是主行，应该尽量把出现频率较高的字母(如 e,t,a,o,i)都放到中间一行。但实际情况是，QWERT 式的键盘在设计时为了故意减慢这些键的输入速度，把它们分散到了上、中、下三行，我们在打字时，手要不停地上下移动。有人曾作过统计，使用 QWERTY 键盘，一个熟练的打字员 8 小时内手指移动的距离长达 25.7 公里，一天下来疲惫不堪。不经常使用的字母 j，k(排在倒数第四和第五位)占据了主行的两个位置，而比较经常使用的 m 和 n 却设置在了最下面一行不显眼的位置。从左右手的工作量来看，我们平常使用的键盘对于左手是不公平的，据统计有 57%的击键由左手完成，而大多数人都不是左撇子。有一些常用词像 was 和 extra 完全要用左手完成，这让打字员的右手可以忙里偷闲，左手却成了“苦劳力”。根据每一只手各个手指的工作量统计，也是不合理的，与手指的力量和灵活性不匹配，例如瘦弱又不灵活的小手指经常受“欺负”，承担的负荷过大。看来，QWERT 式的键盘似乎是一个失败的键盘设计。1936 年美国人 Dvorak 根据以上的这些研究发明了一种新型的键盘。

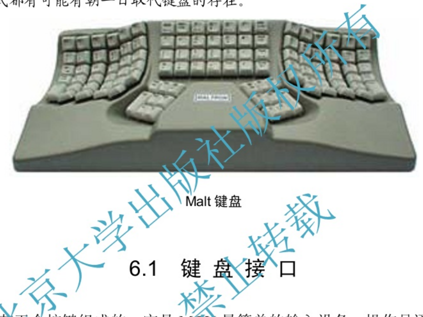


Dvorak 键盘

在这种键盘上，可以看到，在主行，AOEUIDHTNS 都是字母频率使用表中排在前列的，而最下面的一行的那些字母都是较少使用的。按照 Dvorak 的解释，70%的按键都可以单靠主行完成，另外 22%的按键靠最上面一行，只有 8%在最下面一行，可以使手指不用总是上下换来换去，符合人的正常习惯。Dvorak 键盘还可以使左右手、各个手指之间的任务量分配更加公平，右手的平均使用时间也超过了左手，不再出现左撇子现象。

在 20 世纪 70 年代，一位名为 Lilian Malt 的发明家又对 Dvorak 键盘作了进一步改进，不仅考虑了字母位置的排列，还将键盘做成弯曲的形状，分为左右两部分，分别由两只手控制，这种设计可以使打字员在打字时身体保持舒服的姿势，手腕不容易酸痛和损伤。

尽管 Dvorak 键盘和 Malt 键盘在易学性、输入速度、人体保健等方面都好于 QWERTY 键盘，当时很多人也乐观地预计它们有很大发展潜力，会很快取代现有键盘。但是实际情况却是，时至今日，计算机前的键盘仍然是 QWERTY 键盘，Dvorak 键盘和 Malt 键盘"出师未捷身先死"，完全没有走入市场。人们普遍认为，QWERTY 键盘作为过时的东西仍然活跃在舞台上的原因主要在于它的先入为主，尽管存在种种缺陷，但是成千上万的使用者已经熟练使用它，加上产品已经成型，电脑从业者也不希望费力地改变与键盘相关的各种硬件软件，引入新的键盘。不过，QWERTY 键盘也不会永存于世，语音识别、手写输入、触摸、点击输入和其他各种更先进的输入方式都有可能有朝一日取代键盘的存在。



Malt 键盘

6.1 键盘接口

键盘是由若干个按键组成的，它是 MCU 最简单的输入设备。操作员通过键盘输入数据或命令，实现简单的人机对话。

键盘是一组按键的集合，按键是一种常开型按钮开关，平时(常态)按键的两个触点处于断开状态，按下按键时它们才闭合(短路)。键盘分编码键盘和非编码键盘，按键的识别由专用的硬件译码实现。能产生按键编号或键值的称为编码键盘，如 BCD 码键盘、ASC 码键盘等，而缺少这种键盘编码电路要靠自编软件识别的称为非编码键盘。在 MCU 组成的电路系统及智能化仪器仪表中，用得更多的是非编码键盘，本节只讨论非编码键盘。

6.1.1 键盘的工作原理

当按键未按下(即断开)时，P1.i(i=0～7)输入为高电平，按键闭合后，P1.i 输入为低电平。通常，按键所用的开关为机械弹性开关，当机械触点断开、闭合时，电压信号波形如图 6-1(a)所示。由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会马上断开，因而在闭合及断开的瞬间均伴随有一连串的抖动。抖动时间的长短由按键的机械特性决定，一般为 5～10ms。因为 MCU 处理的速度在微秒级，这种抖



动对于人来说是感觉不到的，但对 MCU 来说，则是完全可以检测到的。假如对按键不进行去抖处理，例如，通过键盘输入一个数字“1”，MCU 程序却已执行了多次输入数字“1”的按键处理程序，其结果是认为我们输入了若干个数字“1”。

按键抖动会引起一次按下按键被误读为多次，为了确保 MCU 对按键的一次闭合仅做一次处理，必须消除按键抖动，在按键闭合稳定时取按键状态，并且必须判别到按键释放稳定后再进行处理。按键的抖动，可用硬件或软件两种方法消除。

通常在按键数较少时，可用硬件方法消除按键抖动。RS 触发器为常用的硬件去抖电路，如图 6-1(b)所示的。MCU 中常用软件法。软件去抖法很简单，就是在 MCU 获得 P1.i 口为低电平的信息后，不是立即认定按键已被按下，而是延时 10ms 或更长一段时间后，再次检测 P1.i 口，如果仍为低电平，说明按键 S 的确被按下了，这实际上是避开了按键被按下时的抖动时间。而在检测到按键释放后(P1.i 为高电平)再延时 5～10ms，消除后沿的抖动，再对键值处理。一般情况下，不对按键释放的后沿进行处理，实践证明，这也能满足一定的要求。当然，实际应用中，对按键的要求千差万别，这就要根据不同的需要编制处理程序。

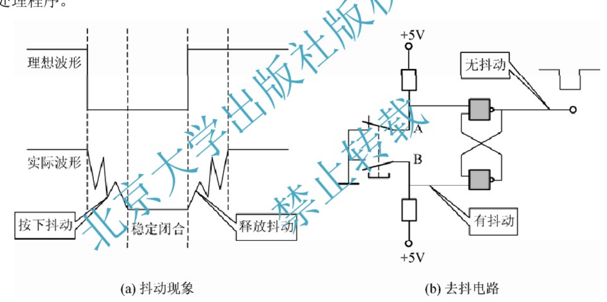


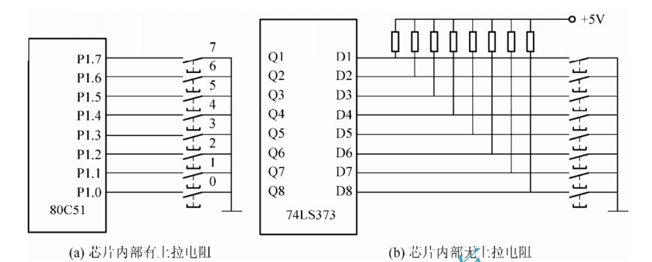
图 6-1　按键的抖动及其消除电路

6.1.2　键盘的工作方式

1. 独立式键盘及其接口

独立式键盘的各个按键相互独立，每个按键独立地与一根数据输入线(MCU 并行口或其他接口芯片的并行口)相连，如图 6-2 所示。

图 6-2(a)为芯片内部有上拉电阻的接口；图 6-2(b)为芯片内部无上拉电阻的接口，这时就应在芯片外设置上拉电阻。独立式键盘配置灵活，软件结构简单，但每个按键必须占用一根口线，在按键数量较多时，口线占用较多，所以独立式按键常用于按键数量不多的场合。

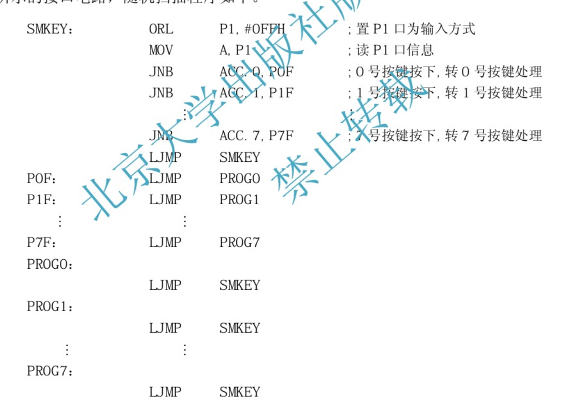


(a) 芯片内部有上拉电阻　　　　(b) 芯片内部无上拉电阻

图 6-2　独立式键盘接口

独立式键盘的软件可以采用随机扫描，也可以采用定时扫描，还可以采用中断扫描。

随机扫描是指当 CPU 空闲时调用键盘扫描子程序，响应键盘的输入请求。对图 6-2(a)所示的接口电路，随机扫描程序如下。



```
SMKEY:      ORL     P1, #0FFH       ; 置 P1 口为输入方式
            MOV     A, P1           ; 读 P1 口信息
            JNB     ACC.0, P0F      ; 0 号按键按下，转 0 号按键处理
            JNB     ACC.1, P1F      ; 1 号按键按下，转 1 号按键处理
              ⋮                       ⋮
            JNB     ACC.7, P7F      ; 7 号按键按下，转 7 号按键处理
            LJMP    SMKEY
P0F:        LJMP    PROG0
P1F:        LJMP    PROG1
  ⋮           ⋮
P7F:        LJMP    PROG7
PROG0:
            LJMP    SMKEY
PROG1:
            LJMP    SMKEY
  ⋮           ⋮
PROG7:
            LJMP    SMKEY
```


定时扫描方式是利用 MCU 内部定时器产生定时中断，在中断服务程序中对键盘进行扫描，并在有按键按下时转入按键功能处理程序。定时扫描方式的硬件接口电路与随机扫描方式相同。

对于中断扫描方式，当键盘上有按键闭合时产生中断请求，CPU 响应中断并在中断服务程序中判别键盘上闭合按键的键号，并作相应的处理。中断扫描接口电路如图 6-3 所示。



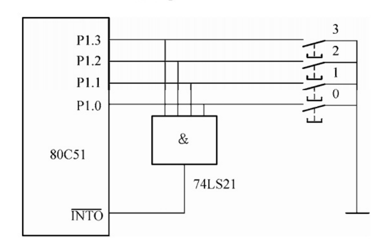


图 6-3　中断扫描接口电路

2. 矩阵式键盘及其接口

矩阵式键盘采用行列式结构，按键设置在行列的交叉点上。当口线数量为 8 时，可以将 4 根口线定义为行线，另外 4 根口线定义为列线，形成 4×4 键盘，可以配置 16 个按键，如图 6-4(a)所示，图 6-4(b)所示为 4×8 键盘。

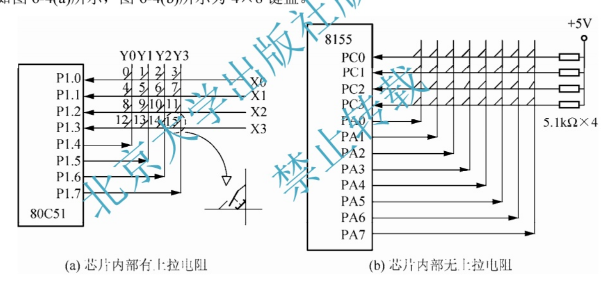


图 6-4　矩阵式键盘

矩阵式键盘的行线通过电阻接+5V(芯片内部有上拉电阻时，就不用外接了)，当键盘上没有按键闭合时，所有的行线与列线是断开的，行线均呈高电平。

当键盘上某一按键闭合时，该按键所对应的行线与列线短接。此时该行线的电平将由被短接的列线电平所决定。因此，可以采用以下方法完成是否有按键按下及按下的是哪一个按键的判断：

(1) 判断有无按键按下。将行线接至 MCU 的输入口，列线接至 MCU 的输出口。首先使所有列线为低电平，然后读行线状态，若行线均为高电平，则没有按键按下；若读出的行线状态不全为高电平，则可以断定有按键按下。

(2) 判断按下的是哪一个按键。先让 Y0 这一列为低电平，其余列线为高电平，读行线状态，如行线状态不全为“1”，则说明所按下的按键在该列，否则不在该列。然后让 Y1 列为低电平，其他列为高电平，判断 Y1 列有无按键按下，其余列类推，这样就可以找到所按下的按键的行列位置。对于图 6-4(a)所示的接口电路，示例程序如下。

```
SMKEY:    MOV     P1, #0FH          ; 置 P1 口高 4 位为“0”、低 4 位为输入状态
          MOV     A, P1             ; 读 P1 口
          ANL     A, #0FH           ; 屏蔽高 4 位
          CJNE    A, #0FH, HKEY     ; 有按键按下, 转 HKEY
          SJMP    SMKEY             ; 无按键按下转回
HKEY:     LCALL   DELAY10           ; 延时 10ms, 去抖
          MOV     A, P1             ;
          ANL     A, #0FH           ;
          CJNE    A, #0FH, WKEY     ; 确认有按键按下, 转到判断哪一按键按下
          SJMP    SMKEY             ; 是抖动转回
WKEY:     MOV     P1, #11101111B    ; 置扫描码, 检测 P1. 4 列
          MOV     A, P1             ;
          ANL     A, #0FH           ;
          CJNE    A, #0FH, PKEY     ; P1. 4 列(Y0) 有按键按下, 转按键处理
          MOV     P1, #11011111B    ; 置扫描码, 检测 P1. 5 列
          MOV     A, P1             ;
          ANL     A, #0FH
          CJNE    A, #0FH, PKEY     ; P1. 5 列(Y1) 有按键按下, 转按键处理
          MOV     P1, #10111111B    ; 置扫描码, 检测 P1. 6
          MOV     A, P1             ;
          ANL     A, #0FH           ;
          CJNE    A, #0FH, PKEY     ; P1. 6 列(Y2) 有按键按下, 转按键处理
          MOV     P1, #01111111B    ; 置扫描码, 检测 P1. 7 列
          MOV     A, P1             ;
          ANL     A, #0FH           ;
          CJNE    A, #0FH, PKEY     ; P1. 7 列(Y3) 有按键按下, 转按键处理
          LJMP  SMKEY               ;
PKEY:                               ; 按键处理
```

执行该程序后，可以获得按下按键所在的行列位置，此种按键识别方法称为扫描法。从原理上易于理解，但当所按按键在最后一列时，所需扫描次数较多。

还可以采用线反转法完成所按按键的识别。先把列线置成低电平，行线置成输入状态，读行线；再把行线置成低电平，列线置成输入状态，读列线。当有按键按下时，由两次所读状态即可确定所按按键的位置，示例程序如下。



```
SMKEY:    MOV     P1,#0FH        ;置P1口高4位为“0”、低4位为输入状态
          MOV     A,P1           ;读P1口
          ANL     A,#0FH         ;屏蔽高4位
          CJNE    A,#0FH,HKEY    ;有按键按下,转HKEY
          SJMP    SMKEY          ;无按键按下转回
HKEY:     LCALL   DELAY10        ;延时10ms,去抖
          MOV     A,P1           ;
          ANL     A,#0FH         ;
          MOV     B,A            ;行线状态在B的低4位
          CJNE    A,#0FH,WKEY    ;确认有按键按下,转判哪一按键按下
          SJMP    SMKEY          ;是抖动转回
WKEY:     MOV     P1,#0F0H       ;置P1口高4位为输入、低4位为“0”
          MOV     A,P1           ;
          ANL     A,#0F0H        ;屏蔽低4位
          ORL     A,B            ;列线状态在高4位,与行线状态合成于B中
          …       …              ;按键处理
```

按键处理是根据所按按键散转进入相应的功能程序。为了散转的方便，通常应先得到按下按键的键号。键号是键盘的每个按键的编号，可以是十进制或十六进制。键号一般通过键盘扫描程序取得的键值求出。键值是各按键所在行号和列号的组合码。如图 6-4(a)所示接口电路中的按键“9”所在行号为 2，所在列号为 1，键值可以表示为“21H”，也可以表示为‘12H’，表示方法并不是唯一的，要根据具体按键的数量及接口电路而定。根据键值中行号和列号信息就可以计算出键号：

键号=所在行号×键盘列数+所在列号

即　　$2\times4+1=9$

根据键号就可以方便地通过散转进入相应按键的功能程序。

6.1.3　键盘的接口电路

在 MCU 的应用系统中，往往将键盘和显示电路合并考虑，从而使接口芯片的利用率提高，系统的设计简化。常用的键盘和显示接口电路有以下几种。

(1) 利用 8155 构成的键盘及显示接口电路。

(2) 利用 8279 构成的键盘及显示接口电路。

(3) 利用 MCU 的串行口构成的键盘及显示接口电路。

1. 8155 的键盘及显示接口

由于 8155 接口芯片含有 MCU 应用系统扩展常用的资源，所以可以方便地利用 8155 构成键盘和显示接口电路，如图 6-5 所示。

图中 6 个数码管显示器采用共阴极的 LED，段数据由 8155 的 B 口提供，位选信号由 A 口提供。键盘的列扫描输出也由 A 口提供，键盘的行输入由 C 口提供。

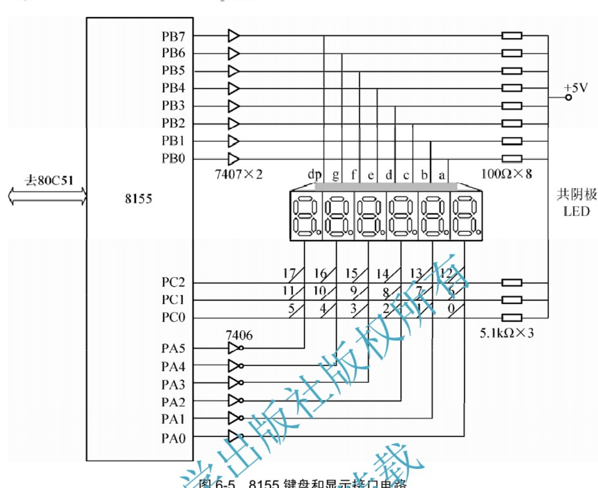


图 6-5　8155 键盘和显示接口电路

实现程序如下。

```
KD1:    MOV     A, #00000011B     ; 8155 初始化: PA、PB 为基本输出, PC 为输入
        MOV     DPTR, #7F00H      ;
        MOV     X  @DPTR, A       ;
KEY1:   ACALL   KS1               ; 查有无键按下
        JNZ     LK1               ; 有, 转按键扫描
        ACALL   DI S              ; 调显示子程序
        AJMP    KEY1              ;
LK1:    ACALL   DI S              ; 按键扫描
        ACALL   DI S              ; 两次调显示子程序, 延时 12ms
        ACALL   KS1               ;
        JNZ     LK2               ;
        ACALL   DI S              ; 调显示子程序
        AJMP    KEY1              ;
LK2:    MOV     R2, #0FEH         ; 从首列开始
        MOV     R4, #00H          ; 首列号送 R4
LK4:    MOV     DPTR, #7F01H      ;
```



```
          MOV     A, R2              ;
          MOVX    A, @DPTR           ;
          INC     DPTR               ;
          INC     DPTR               ; 指向 C 口
          MOVX    @DPTR, A           ;
          JB      ACC.0, LONE        ; 第 0 行无按键按下, 转查第 1 行
          MOV     A, #00H            ; 第 0 行有按键按下, 该行首键号送 A
          AJMP    LKP                ; 转求键号
LONE:     JB      ACC.1, LTWO        ; 第 1 行无按键按下, 转查第 2 行
          MOV     A, #08H            ; 第 1 行有按键按下, 该行首键号送 A
          AJMP    LKP                ; 转求键号
LTWO:     JB      ACC.2, NEXT        ; 第 2 行无按键按下, 转查下一列
          MOV     A, #10H            ; 第 2 行有按键按下, 该行首键号送 A
LKP:      ADD     A, R4              ; 求键号。键号=行首键号+列号
          PUSH    ACC                ; 保护键号
LK3:      ACALL   DIS                ; 等待按键释放
          ACALL   KS1                ;
          JNZ     LK3                ;
          POP     ACC                ;
          RET                        ; 按键扫描结束。此时 A 的内容为按下按键的键号
NEXT:     INC     R4                 ; 指向下一列
          MOV     A, R2              ;
          JNB     ACC.5, KND         ; 判断 6 列扫描完没有
          RL      A                  ; 未完, 扫描字对应下一列
          MOV     R2, A              ;
          AJMP    LK4                ; 转下一列扫描
KND:      AJMP    KEY1               ; 扫描完, 转入新一轮扫描
KS1:      MOV     DPTR, #7F01H       ; 查有无按键按下子程序。先指向 A 口
          MOV     A, #00H            ;
          MOVX    @DPTR, A           ; 送扫描字“00H”
          INC     DPTR               ;
          INC     DPTR               ; 指向 C 口
          MOVX    A, @DPTR           ;
          CPL     A                  ; 变正逻辑
          ANL     A, #0FH            ; 屏蔽高位
          RET                        ; 子程序出口, A 的内容非 0 则有按键按下
```


2. 8279 的键盘及显示接口

8279 作为键盘/显示的专用接口芯片，在键盘/显示接口电路的设计中具有明显的优势。图 6-6 所示为 8279 的典型用法。

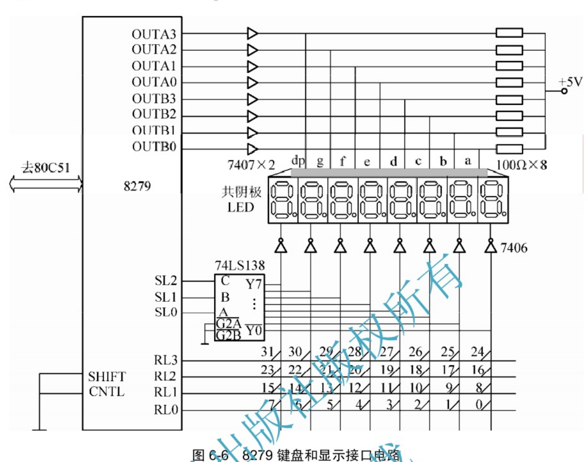


图 6-6 8279 键盘和显示接口电路

初始化程序如下。

```
INIT:   MOV     DPTR, #7FFFH    ;置 8279 命令/状态口地址
        MOV     A, #0D1H        ;置清显示命令字
        MOVX    @DPTR, A        ;送清显示命令
WEIT:   MOVX    A, @DPTR        ;读状态
        JB      ACC.7, WEIT     ;等待清显示 RAM 结束
        MOV     A, #34H         ;置分频系数,晶振频率 12MHz
        MOVX    @DPTR, A        ;送分频系数
        MOV     A, #40H         ;置键盘/显示命令
        MOVX    @DPTR, A        ;送键盘/显示命令
        MOV     IE, #84H        ;允许 8279 中断
        RET
```

显示子程序如下。

```
DIS:    MOV     DPTR, #7FFFH    ;置 8279 命令/状态口地址
        MOV     R0, #30H        ;字段码首地址
        MOV     R7, #08H        ;8 位显示
        MOV     A, #90H         ;置显示命令字
        MOVX    @DPTR, A        ;送显示命令
        MOV     DPTR, #7FFEH    ;置数据口地址
```



```
LP:     MOV     A,@R0           ;取显示数据
        ADD     A,#5            ;加偏移量
        MOVC    A,@A+PC         ;查表,取得数据的段码
        MOVX    @DPTR,A         ;送段码显示
        INC     R0              ;调整数据指针
        DJNZ    R7,LP           ;
        RET                     ;
        SEG:    DB      3FH,06H,5BH,4FH,66H,6DH
                                ;字符 0、1、2、3、4、5 段码
                DB      7DH,07H,7EH,6FH,77H,7CH
                                ;字符 6、7、8、9、A、b 段码
                DB      39H,5EH,79H,71H,73H,3EH
                                ;字符 C、d、E、F、P、U 段码
                DB      76H,38H,40H,6EH,FFH,00H
                                ;字符 H、L、-、Y、日、"空"段码
```

键盘中断子程序如下。

```
KEY:    PUSH    PSW
        PUSH    DPL
        PUSH    DPH
        PUSH    ACC
        PUSH    B
        SETB    PSW.3
        MOV     DPTR,#7FFFH     ;置状态口地址
        MOVX    A,@DPTR         ;读 FIFO 状态
        ANL     A,#0FH          ;
        JZ      PKYR            ;
        MOV     A,#40H          ;置读 FIFO 命令
        MOVX    @DPTR,A         ;送读 FIFO 命令
        MOV     DPTR,#7FFEH     ;置数据口地址
        MOVX    A,@DPTR         ;读数据
        LJMP    KEY1            ;转键值处理程序
PKYR:   POP     B
        POP     ACC
        POP     DPH
        POP     DPL
        POP     PSW
        RETI                    ;
KEY1:                           ;键值处理程序
```


3. 串行口键盘及显示接口电路

当 80C51 系列 MCU 的串行口未用于串行通信时，可以将其用于键盘和显示器的接口



扩展，这里仅给出接口电路，如图 6-7 所示。

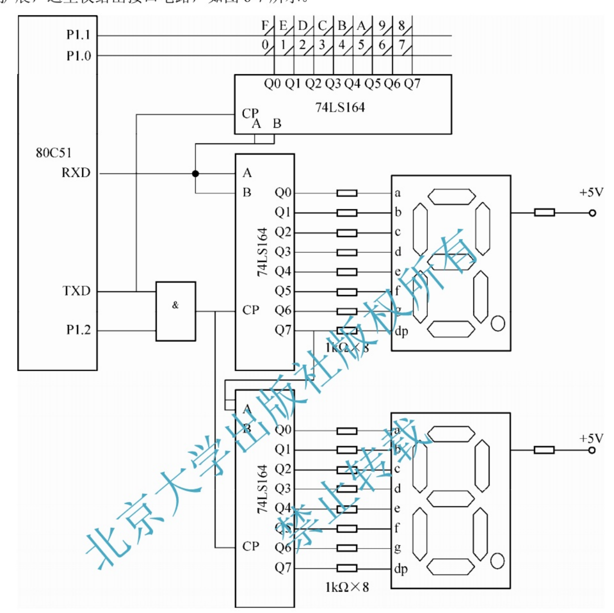


图 6-7　串行口键盘及显示接口电路

6.2　显 示 接 口

在 MCU 应用系统中，为了便于人们观察和监视 MCU 的运行情况，常常需要用显示器显示运行的中间结果及状态等信息，因此显示器也是不可缺少的外部设备之一。

显示器的种类很多，从液晶显示器、发光二极管显示器到 CRT 显示器，都可以与 MCU 配接。在 MCU 应用系统中常用的显示器主要有发光二极管数码显示器(简称 LED 显示器)和液晶显示器(简称 LCD 显示器)。LED 显示器具有耗电省、成本低廉、配置简单灵活、安装方便、耐振动、寿命长等优点。但显示内容有限，且不能显示图形，因而其应用有局限性；LCD 显示器除了具有 LED 的特点外还能实行图形显示，但其驱动较为复杂。



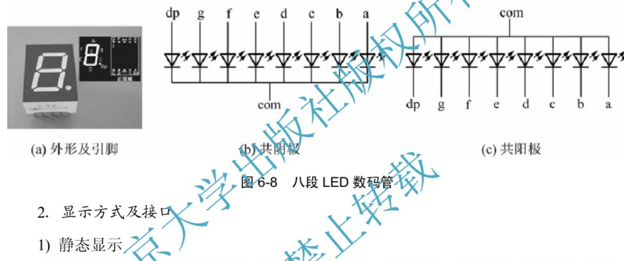
本节着重介绍 LED、LCD 显示器工作原理。

6.2.1　LED 显示器

1. 八段显示器的原理

显示器是 MCU 应用系统常用的设备，包括 LED、LCD 等。LED 显示器由若干个发光二极管组成，当发光二极管导通时，相应的一个笔画或一个点就发光。控制相应的二极管导通，就能显示出对应字符。八段 LED 显示器如图 6-8 所示。

八段 LED 通常用七段构成字形“8”，还有一个发光二极管用来显示小数点。各段 LED 显示器需要由驱动电路驱动。在八段 LED 显示器中，通常将各段发光二极管的阴极或阳极连在一起作为公共端，这样可以使驱动电路简单。将各段发光二极管阳极连在一起的叫共阳极显示器，用低电平驱动；将阴极连在一起的叫共阴极显示器，用高电平驱动。



(a) 外形及引脚　　(b) 共阴极　　(c) 共阳极

图 6-8　八段 LED 数码管

2. 显示方式及接口

1) 静态显示

所谓静态显示，是指显示器显示某一字符时，相应段的发光二极管恒定地导通或截止。静态显示有并行输出和串行输出两种方式。

图 6-9 所示为并行输出的静态显示电路。八段 LED 显示器的 a、b、c、d、e、f 段导通，g、dp 段截止，则显示“0”。并行显示方式每个十进制位都需要有一个 8 位输出口控制，图中采用 3 片 74LS373 扩展并行 I/O 口，口地址是由 74LS138 译码器的输出决定的，74LS138 的 A、B、C 分别接 80C51 的 P2.5、P2.6 和 P2.7，所以 3 片 74LS373 的地址分别为 1FFFH、3FFFH、5FFFH。译码输出信号与 MCU 的写信号一起控制对各 74LS373 数据的写入。

对于静态显示方式，LED 显示器由接口芯片直接驱动，采用较小的驱动电流就可以得到较高的显示亮度。但是，并行输出显示的十进制位数多时，需要并行 I/O 接口芯片的数量较多。

采用串行输出可以大大节省 MCU 的内部资源。图 6-10 为串行输出的静态显示电路。串并转换器采用 74LS164，低电平时允许通过 8mA 电流，不需要添加其他驱动电路。TXD 为移位时钟输出，RXD 为移位数据输出，P1.0 作为显示器允许控制输出线。每次串行输出 24 位(3 个字节)的段码数据。

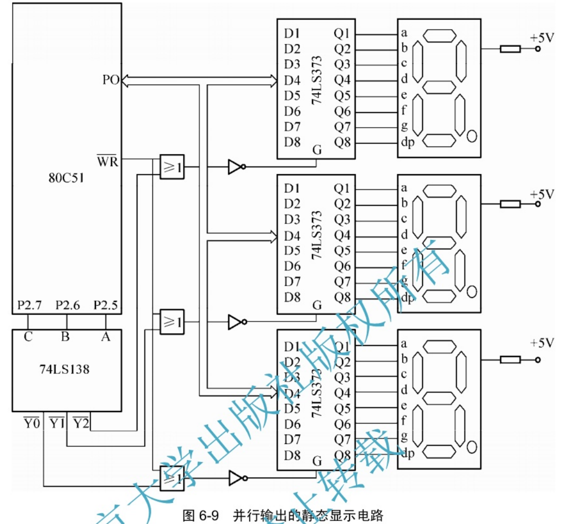


图 6-9 并行输出的静态显示电路

2) 动态显示

当显示器位数较多时，可以采用动态显示。所谓动态显示，就是一位一位地轮流点亮显示器的各个位(扫描)。对于显示器的每一位而言，每隔一段时间点亮一次。虽然在同一时刻只有一位显示器在工作(点亮)，但由于人眼的视觉暂留效应和发光二极管熄灭时的余光，我们看到的却是多个字符“同时”显示。显示器亮度既与点亮时的导通电流有关，也与点亮时间长短和间隔时间有关。调整电流和时间参数，即可实现亮度较高且较稳定的显示。

若显示器的位数不大于 8 位，则控制显示器公共极电位只需一个 I/O 口(称为扫描口或字位口)，控制各位 LED 显示器所显示的字形也需要一个 8 位口(称为段数据口或字形口)。图 6-11 为位动态 LED 显示接口。8155 的端口 A 作为扫描口(字位口)，经反相驱动器 7406 接显示器公共极。端口 B 作为段数据口(字形口)，经同相驱动器 7407 接显示器的各个极。

对应图中的 6 位 LED 显示器，MCU 内部 RAM 中设置了 6 个显示缓冲单元 79H～7EH，存放 6 位将要显示的字符数据。8155 的端口 A 扫描输出总是有一位为高电平，以选中相应的字位。端口 B 输出相应位的显示字符段数据，使该位显示出相应字符(点亮)，其他位为熄灭状态。依次改变端口 A 输出为高电平的位及端口 B 输出对应的段数据，6 位 LED 显示器就可以显示出缓冲器中字符数据所确定的字符。



图 6-10　串行输出的静态显示电路

图 6-11　位动态 LED 显示接口



程序清单如下。

```
DIS:     MOV    R0, #79H        ; 显示数据缓冲区首地址送 R0
         MOV    R3, #01H        ; 使显示器最右边位亮
         MOV    A, R3           ;
LD0:     MOV    DPTR, #7F01H    ; 数据指针指向 A 口
         MOVX   @DPTR, A        ; 送扫描值
         INC    DPTR            ; 数据指针指向 B 口
         MOV    A, @R0          ; 取欲显示的数据
         ADD    A, #0DH         ; 加上偏移量
         MOVC   A, @A+PC        ; 取出字形码
         MOVX   @DPTR, A        ; 送显示
         ACALL  DL1             ; 调用延时子程序
         INC    R0              ; 指向下一个显示段数据地址
         MOV    A, R3           ;
         JB     ACC.5, ELD1     ; 扫描到第 6 个显示器否?
         RL     A               ; 未到, 扫描码左移 1 位
         MOV    R3, A
         AJMP   LD0
ELD1:    RET
DSEG:    DB     3FH, 06H, 5BH, 4FH, 66H, 6DH
         DB     7DH, 07H, 7FH, 6FH, 77H, 7CH
         DB     39H, 5EH, 79H, 71H, 40H, 00H
DL1:     MOV    R7, #02H        ; 延时 1ms 子程序
DL:      MOV    R6, #0FFH
DL6:     DJNZ   R6, DL6
         DJNZ   R7, DL
         RET
```


若某些字符的显示需要带小数点(DP)或需要数据的某些位闪烁时(点亮一段时间，熄灭一段时间)，则可建立小数点位置及数据闪烁位置标志单元，指出小数点显示位置和闪烁位置。当显示扫描到相应位时(字位选择字与小数点位置字或闪烁位置字重合)，加入小数点或控制该位闪烁，完成带小数点或闪烁字符的显示。

6.2.2 LCD 显示器

液晶显示器简称 LCD(Liquid Crystal Display)。它是一种被动式的显示器，与 LED 数码显示器不同，液晶本身并不发光，而是利用液晶电压作用下能改变光线通过方向的特性，而达到显示白底黑字或黑底白字的目的。液晶显示器具有体积小、功耗极低、抗干扰能力强、显示内容丰富等特点，在 MCU 应用系统中有着日益广泛的应用。

常见的液晶显示器有八段式 LCD、点阵式字符型 LCD 和点阵式图形 LCD，下面重点介绍点阵式字符型 LCD。



1. 点阵式字符型 LCD 的结构和工作原理

字符型液晶显示模块是专门用于显示字母、数字、符号等的点阵型液晶显示模块。分 4 位和 8 位数据传输方式。提供 5×7 点阵+光标和 5×10 点阵+光标的显示模式。提供内部上电自动复位电路，当外加电源电压超过+4.5V 时，自动对模块进行初始化操作，将模块设置为默认的显示工作状态。

字符型液晶显示模块组件内部主要由 LCD 显示屏(LCD PANEL)、控制器(controller)、驱动器(driver)、少量电阻电容元件、结构件等装配在 PCB 板上构成，如图 6-12 所示。

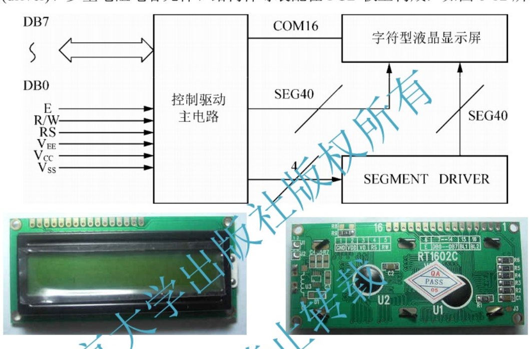


图 6-12　字符型液晶显示模块及 PCB 板

字符型液晶显示模块目前在国际上已经规范化，无论显示屏规格如何变化，其电特性和接口形式都是统一的。因此只要设计出一种型号的接口电路，在指令设置上稍加改动即可使用各种规格的字符型液晶显示模块。

字符型液晶显示模块的基本特点如下。

(1) 液晶显示屏是以若干个 5×8 或 5×11 点阵块组成的显示字符群。每个点阵块为一个字符位，字符间距和行距都为一个点的宽度。

(2) 主控制驱动电路为 HD44780 以及其他公司全兼容电路，如 SED1278、KS0066 和 NJU6408。

(3) 具有字符发生器 ROM，可显示 192 种字符。

(4) 具有 64 个字节的自定义字符 RAM，可自定义 8 个 5×8 点阵字符和 4 个 5×11 点阵字符。

(5) 具有 80 个字节的 RAM。

(6) 标准的接口特性，适配 M6800 系列 MPU 的操作时序。

(7) 模块结构紧凑、轻巧、装配简单。

(8) 单+5V 电源供电。

(9) 低功耗、长寿命、高可靠性。

下面介绍点阵式字符型 LCD 液晶显示器 LCD1602 模块及其应用。

2. LCD1602 模块接口引脚功能

LCD1602 共有 16 个引脚见表 6-1。

表 6-1　LCD1602 共有 16 个引脚

编号	符号	引脚说明	编号	符号	引脚说明
1	VSS	电源地	9	D2	数据
2	VDD	电源正极	10	D3	数据
3	VL	液晶显示偏压	11	D4	数据
4	RS	数据/命令选择	12	D5	数据
5	R/W	读/写选择	13	D6	数据
6	E	使能信号	14	D7	数据
7	D0	数据	15	BLA	背光源正极
8	D1	数据	16	BLK	背光源负极

管脚进一步说明如下。

(1) 第 1 脚：VSS 接电源地。

(2) 第 2 脚：VDD 接 5V 正电源。

(3) 第 3 脚：VL 为液晶显示器对比度调整端，接正电源时对比度最弱，接地时对比度最高，对比度过高时会产生“鬼影”，使用时可以通过一个 10kΩ的电位器调整对比度。

(4) 第 4 脚：RS 为寄存器选择，高电平时选择数据寄存器，低电平时选择指令寄存器。

(5) 第 5 脚：R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 R/W 共同为低电平时可以写入指令或者显示地址，当 RS 为低电平 R/W 为高电平时可以读忙信号，当 RS 为高电平 R/W 为低电平时可以写入数据。

(6) 第 6 脚：E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。

(7) 第 7～14 脚：D0～D7 为 8 位双向数据线。

(8) 第 15 脚：背光源正极。

(9) 第 16 脚：背光源负极。

3. LCD1602 的指令说明及时序

1602 液晶模块内部的控制器共有 11 条控制指令见表 6-2。



表 6-2　控制指令表

序号	指　　令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	—
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移动	0	0	0	0	0	1	S/C	R/L	—	—
6	置功能	0	0	0	0	1	DL	N	F	—	—
7	置字符发生存储器地址	0	0	0	1	字符发生存储器地址					
8	置数据存储器地址	0	0	1	显示数据存储器地址						
9	读忙标志或地址	0	1	BF	计数器地址						
10	写数到 CGRAM 或 DDRAM	1	0	要写的数据内容							
11	从 CGRAM 或 DDRAM 读数	1	1	读出的数据内容							

1602 液晶模块的读写操作、屏幕和光标的操作都是通过指令编程来实现的(说明：1 为高电平、0 为低电平)。

指令 1：清显示，指令码 01H，光标复位到地址 00H 位置。

指令 2：光标复位，光标返回到地址 00H。

指令 3：光标和显示模式设置。I/D 为光标移动方向，高电平右移，低电平左移；S 为屏幕上所有文字是否左移或者右移，高电平表示有效，低电平则无效。

指令 4：显示开/关控制。D 为控制整体显示的开与关，高电平表示开显示，低电平表示关显示；C 为控制光标的开与关，高电平表示有光标，低电平表示无光标；B 为控制光标是否闪烁，高电平闪烁，低电平不闪烁。

指令 5：光标或显示移位。S/C 表示高电平时移动显示的文字，低电平时移动光标。

指令 6：功能设置命令。DL 表示高电平时为 4 位总线，低电平时为 8 位总线；N 表示低电平时为单行显示，高电平时双行显示；F 表示低电平时显示 5×7 的点阵字符，高电平时显示 5×10 的点阵字符。

指令 7：字符发生器 RAM 地址设置。

指令 8：DDRAM 地址设置。

指令 9：读忙标志和光标地址。BF 为忙标志位，高电平表示忙，此时模块不能接收命令或者数据，如果为低电平则表示不忙。

指令 10：写数据。

指令 11：读数据。

与 HD44780 相兼容的芯片时序表见表 6-3。

表 6-3　基本操作时序表

读状态	输入	RS=L，R/W=H，E=H	输出	D0～D7=状态字
写指令	输入	RS=L，R/W=L，D0-D7=指令码，E=高脉冲	输出	无
读数据	输入	RS=H，R/W=H，E=H	输出	D0～D7=数据
写数据	输入	RS=H，R/W=L，D0～D7=数据，E=高脉冲	输出	无

读写操作时序如图 6-13 和图 6-14 所示。

图 6-13　读操作时序

图 6-14　写操作时序



4. LCD1602 的 RAM 地址映射

液晶显示模块是一个慢显示器件，所以在执行每条指令之前一定要确认模块的忙标志为低电平，表示不忙，否则此指令失效。要显示字符时要先输入显示字符地址，也就是告诉模块在哪里显示字符，图 6-15 是 LCD1602 的内部显示地址。

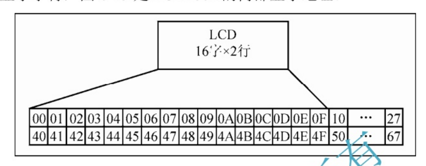


图 6-15　LCD1602 内部显示地址

例如第二行第一个字符的地址是 40H，那么是否直接写入 40H 就可以将光标定位在第二行第一个字符的位置呢？这样不行，因为写入显示地址时要求最高位 D7 恒定为高电平“1”，所以实际写入的数据应该是 01000000B(40H)+10000000B(80H)=11000000B(C0H)。

在对液晶模块的初始化中要先设置其显示模式，在液晶模块显示字符时光标是自动右移的，不需要人工干预。每次输入指令前都要判断液晶模块是否处于忙的状态。

LCD1602 液晶模块内部的字符发生存储器(CGROM)已经存储了 160 个不同的点阵字符图形，这些字符有：阿拉伯数字、英文字母的大小写、常用的符号和日文假名等，每一个字符都有一个固定的代码，比如大写的英文字母“A”的代码是 01000001B(41H)，显示时模块把地址 41H 中的点阵字符图形显示出来，我们就能看到字母“A”。

5. LCD1602 与 80C51 系列 MCU 接口

LCD1602 适配 M6800 系列 MPU 的操作时序，可直接与该系列 MPU 连接。由于 80C51 系列 MCU 的操作时序与 M6800 系列不同，可采用 80C51 的 I/O 模拟 LCD1602 的操作时序，其连接方法如图 6-16 所示。

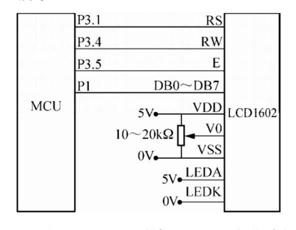


图 6-16　80C51 系列 MCU 的 I/O 模拟 LCD1602 操作时序的接口电路

采用图 6-16 中接口电路将占用很多 80C51 系列 MCU 宝贵的 I/O 口资源，将 80C51 系列 MCU 的读写信号经门电路变换后，可直接将 LCD1602 连到 80C51 系列 MCU 的三总线上。图 6-17 所示为由 AT89C52、74HC00 和 74HC573 组成的接口电路。

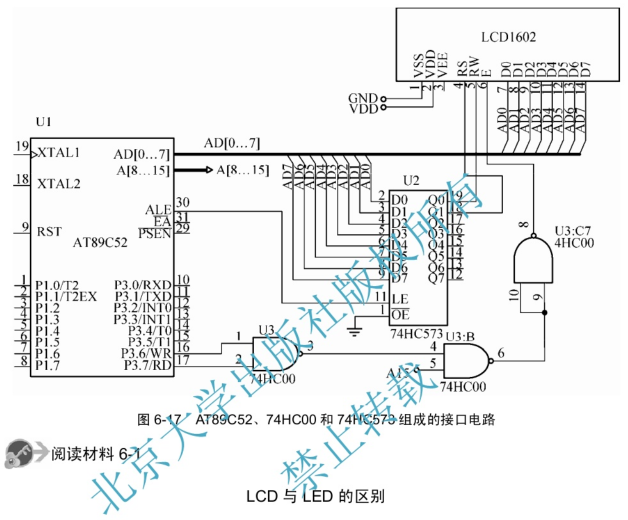


图 6-17　AT89C52、74HC00 和 74HC573 组成的接口电路

阅读材料 6-1

LCD 与 LED 的区别

LCD 是 Liquid Crystal Display 的简称，其构造主要是在两片平行的玻璃当中放置液态的晶体，在这平行的两片玻璃中间有许多垂直和水平的细小电线，通过通电与否来控制杆状水晶分子改变方向，将光线折射出来产生画面。

LED 是 Light Emitting Diode 的简称，它的工作原理是在某些半导体材料的 PN 结中，注入的少数载流子与多数载流子复合时会把多余的能量以光的形式释放出来，从而把电能直接转换为光能；PN 结加反向电压，少数载流子难以注入，则不发光。这种利用注入式电致发光原理制作的二极管叫发光二极管，也就是我们通称的 LED。

LCD 与 LED 最大的区别在于应用了两种不同的显示技术，LCD 是由液态晶体组成的显示屏，而 LED 则是由发光二极管组成的显示屏；其各种主要的区别如下。

(1) 亮度。LED 显示屏的单个元素反应速度是 LCD 液晶屏的 1000 倍，其亮度相对 LCD 液晶屏来说也更加的有优势，LED 显示屏在强光下也可以照看不误，并且适应-40℃的低温。

(2) 功耗。功耗是人们最为关注的一点，LED 的功耗与 LCD 相比，其比值约为 10∶1，在这方面，LED 具有很大的优势。

(3) 可视角度。LCD 显示器的角度限制很大，这是一个很让人头疼的问题，只要偏离的角



度稍大，便无法看到原来的颜色，甚至什么都看不到；而 LED 能提供达到 160°的视角，优势极大。

(4) 刷新速率。LED 利用电能转化为光能，采用注入式原理，因而其刷新速率更高，继而在视屏处理方面更好。

(5) 对比度。目前已知的较高对比度的 LCD 显示屏为 350∶1，但在很多情况下，这无法满足各种需求，但 LED 显示屏却可以达到更高，所以运用更广。

(6) 外观。LED 利用发光二极管为基础，相对 LCD 来说，其显示器能做到更薄。

(7) 使用范围。LED 显示屏的运用范围比 LCD 显示器更广，它可以显示各种文字、数字、彩色图像及动画信息，也可以播放电视、录像、VCD、 DVD 等彩色视频信号，更重要的是可以利用多幅显示屏进行联网播出。

(8) 使用寿命。LED 显示屏一般可支持 10 万小时左右，正常情况下，能使用几十年，而 LCD 的寿命却要短得多。

(9) 色彩。LED 显示屏比 LCD 显示屏的色彩要真实，这是因为 LED 的色域比 LCD 多很多。

6.3　DAC 接口

数/模转换器(D/A 转换器，DAC)是一种把数字信号转换成模拟信号的器件。数字量是二进制代码的位组合，每一位数字代码都有一定的“权”，并对应一定大小的模拟量。为了将数字量转换成模拟量，应将数字量的每一位都转换为相应的模拟量，然后对其求和即可得到与该数字量成正比的模拟量。

6.3.1　D/A 转换器概述

1. D/A 转换器的基本原理及分类

计算机输出的数字信号首先传送到数据锁存器(或寄存器)中，然后由模拟电子开关把数字信号的高低电平变成对应的电子开关状态。当数字量某位为 1 时，电子开关就将基准电压源 V_{REF} 接入电阻网络的相应支路；若为 0 时，则将该支路接地。各支路的电流信号经过电阻网络加权后，由运算放大器求和并变换成电压信号，作为 D/A 转换器的输出。目前常用的数/模转换器是由 T 形电阻网络构成的，一般称其为 T 形电阻网络 D/A 转换器，如图 6-18 所示。

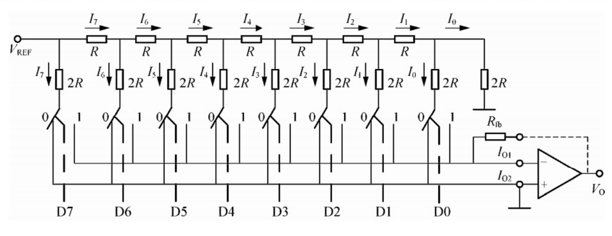


图 6-18　DAC 原理图

该电路是一个 8 位 D/A 转换器，V_{REF} 为外加基准电源，R_{fb} 为外接运算放大器的反馈电阻。D7～D0 为控制电流开关的数据。由图可得

$$I=V_{REF}/R$$

$$I_7=I/2^1，I_6=I/2^2，I_5=I/2^3，I_4=I/2^4，I_3=I/2^5，I_2=I/2^6，I_1=I/2^7，I_0=I/2^8$$

当输入数据 D7～D0 为 11111111B 时，有

$$I_{O1}=I_7+I_6+I_5+I_4+I_3+I_2+I_1+I_0=(I/2^8)\times(2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0)$$

$$I_{O2}=0$$

若 $R_{fb}=R$，则

$$\begin{aligned}V_O&=-I_{O1}\cdot R_{fb}\\&=-I_{O1}\cdot R\\&=-((V_{REF}/R)/2^8)\times(2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0)\times R\\&=-(V_{REF}/2^8)\times(2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0)\end{aligned}$$

由此可见，输出电压 V_O 的大小与数字量具有对应的关系，这样就完成了数字量到模拟量的转换。

D/A 转换器的种类很多，根据数字量的位数划分，有 8 位、10 位、12 位、16 位 D/A 转换器；根据数字量的数码形式划分，有二进制码和 BCD 码 D/A 转换器；根据数字量的传送方式划分，有并行和串行 D/A 转换器；根据 D/A 转换器输出方式划分，有电流输出型和电压输出型 D/A 转换器。

早期的 D/A 转换芯片只具有电流输出型的，且不具有输入寄存器，所以在 MCU 应用系统中使用这种芯片必须外加数字输入锁存器、基准电压源以及输出电压转换电路。这一类芯片主要有 DAC0800 系列(美国 National Semiconductor 公司生产)、AD7520 系列(美国 Analog Devices 公司生产)等。

中期的 D/A 转换芯片在芯片内增加了一些与计算机接口相关的电路及控制引脚，具有数字输入寄存器，能和 CPU 数据总线直接相连。通过控制端，CPU 可直接控制数字量的输入和转换，并且可以采用与 CPU 相同的+5 V 电源供电。这类芯片特别适用于 MCU 应用系统的 D/A 转换接口。这类芯片主要有 DAC0830 系列、AD7524 等。

近期的 D/A 转换器将一些 D/A 转换外围器件集成到了芯片的内部，简化了接口逻辑，提高了芯片的可靠性及稳定性。如芯片内部集成有基准电压源、输出放大器及可实现模拟电压的单极性或双极性输出等。这类芯片主要有 DAC82XX 系列、DAC811、AD558、AD558X 系列等。

2. D/A 转换器的主要性能指标

1) 分辨率

分辨率是指输入数字量的最低有效位(LSB)发生变化时，所对应的输出模拟量(常为电压)的变化量，它反映了输出模拟量的最小变化值。

分辨率与输入数字量的位数有确定的关系，可以表示成 FS/2^n。FS 表示满量程输入值，n 为二进制位数。对于 5V 的满量程，采用 8 位的 D/A 转换器时，分辨率为 5V/2^8=19.5mV；当采用 12 位的 D/A 转换器时，分辨率则为 5V/2^{12}=1.22mV。显然，位数越多分辨率就越高。



2) 线性度

线性度(也称非线性误差)是实际转换特性曲线与理想直线特性之间的最大偏差。常以相对于满量程的百分数表示，如±1%是指实际输出值与理论值之差在满刻度的±1%以内。

3) 绝对精度和相对精度

绝对精度(简称精度)是指在整个刻度范围内，任一输入数码所对应的模拟量实际输出值与理论值之间的最大误差。绝对精度是由 D/A 转换器的增益误差(当输入数码为全 1 时，实际输出值与理想输出值之差)、零点误差(数码输入为全 0 时，D/A 转换器的非零输出值)、非线性误差和噪声等引起的。绝对精度(即最大误差)应小于 1 个 LSB。

相对精度与绝对精度表示同一含义，其用最大误差相对于满刻度的百分比表示。

4) 建立时间

建立时间是指输入的数字量发生满刻度变化时，输出模拟信号达到满刻度值的±1/2LSB 所需的时间，是描述 D/A 转换速率的一个动态指标。

电流输出型 D/A 转换器的建立时间短。电压输出型 D/A 转换器的建立时间主要决定于运算放大器的响应时间。根据建立时间的长短，可以将 D/A 转换器分成超高速(＜1μs)、高速(10～1μs)、中速(100～10μs)、低速(≥100μs)几挡。

应当注意，精度和分辨率具有一定的联系，但概念不同。D/A 转换器的位数多时，分辨率会提高，对应于影响精度的量化误差会减小，但其他误差(如温度漂移、线性不良等)的影响仍会使 D/A 转换器的精度变差。

6.3.2　微控制器与 DAC0832 的接口设计

DAC0832 是使用非常普遍的 8 位 D/A 转换器，由于其片内有输入数据寄存器，故可以直接与 MCU 接口。DAC0832 以电流形式输出，当需要转换为电压输出时，可外接运算放大器。属于该系列的芯片还有 DAC0830、DAC0831，它们可以相互代换。

DAC0832 主要特性如下。

(1) 分辨率 8 位。

(2) 电流建立时间 1μs。

(3) 数据输入可采用双缓冲、单缓冲或直通方式。

(4) 输出电流线性度可在满量程下调节。

(5) 逻辑电平输入与 TTL 电平兼容。

(6) 单一电源供电(+5～+15V)。

(7) 低功耗，20mW。

1. DAC0832 的内部结构及引脚

DAC0832 由一个 8 位输入锁存器、一个 8 位 DAC 寄存器和一个 8 位 D/A 转换器及逻辑控制电路组成。输入数据锁存器和 DAC 寄存器构成了两级缓存，可以实现多通道同步转换输出，如图 6-19 所示。

由图可见，DAC0832 芯片内具有两级输入锁存结构，可以工作于双缓冲、单缓冲和直通方式，使用非常灵活方便。

DAC0832 采用 20 脚双列直插式封装，其引脚如图 6-20 所示。各引脚功能如下。

$\overline{CS}$：片选信号引脚，输入低电平有效。与 ILE 相配合，可对写信号 $\overline{WR1}$ 是否有效起到控制作用。

ILE：允许锁存信号引脚，输入高电平有效。输入锁存器的信号 $\overline{LE1}$ 由 ILE、$\overline{CS}$、$\overline{WR1}$ 的逻辑组合产生。当 ILE 为高电平，$\overline{CS}$ 为低电平，$\overline{WR1}$ 输入低电平时，$\overline{LE1}$ 信号为低电平。$\overline{LE1}$ 为高电平时，输入锁存器的状态随着数据输入线的状态变化，$\overline{LE1}$ 的下降沿将数据线上的信息锁入输入锁存器。

图 6-19　DAC0832 内部逻辑框图

图 6-20　DAC0832 引脚



$\overline{WR1}$：写信号 1 引脚，输入低电平有效。当 $\overline{WR1}$、$\overline{CS}$、ILE 均为有效时，可将数据写入输入锁存器。

$\overline{WR2}$：写信号 2 引脚，输入低电平有效。当其有效时，在传送控制信号 $\overline{XFER}$ 的作用下，可将锁存在输入锁存器的 8 位数据送到 DAC 寄存器。

$\overline{XFER}$：数据传送控制信号引脚，输入低电平有效。当 $\overline{XFER}$ 为低电平，$\overline{WR2}$ 输入负脉冲时，则在 $\overline{LE2}$ 产生正脉冲。$\overline{LE2}$ 为高电平时，DAC 寄存器的输出和输入锁存器状态一致，$\overline{LE2}$ 的负跳变将输入锁存器的内容锁入 DAC 寄存器。

V_{REF}：基准电压输入引脚，可在－10～+10 V 范围内调节。

DI7～DI0：数字量数据输入引脚。

I_{OUT1}、I_{OUT2}：电流输出引脚。电流 I_{OUT1} 与 I_{OUT2} 的和为常数，I_{OUT1}、I_{OUT2} 随寄存器的内容线性变化。

R_{fb}：DAC0832 芯片内部反馈电阻引脚。

VCC：电源输入引脚，+5～+15 V。

DGND、AGND：分别为数字信号地和模拟信号地。

2. DAC0832 与 80C51 系列 MCU 的接口

DAC0832 可工作于单缓冲、双缓冲及直通 3 种方式。

1) 单缓冲工作方式

单缓冲方式，即输入锁存器和 DAC 寄存器相应的控制信号引脚分别连在一起，使数据直接写入 DAC 寄存器，立即进行 D/A 转换(这种情况下，输入锁存器不起锁存作用)。此方式适用于只有一路模拟量输出，或有几路模拟量输出但并不要求同步的系统。

图 6-21 为单极性 1 路模拟量输出的 DAC0832 与 80C51 系列 MCU 接口电路。图中 ILE 接+5V，I_{OUT2} 接地，I_{OUT1} 输出电流经运算放大器变换后输出单极性电压，范围为 0～+5V。片选信号 $\overline{CS}$ 和数据传送控制信号 $\overline{XFER}$ 都与 80C51 的地址线相连(图中为 P2.7)，因此输入锁存器和 DAC 寄存器的地址都为 7FFFH。$\overline{WR1}$、$\overline{WR2}$ 均与 80C51 的写信号线 $\overline{WR}$ 相连。CPU 对 DAC0832 执行一次写操作，则将一个数据直接写入 DAC 寄存器，DAC0832 的输出模拟量随之变化。由于 DAC0832 具有数字量的输入锁存功能，故数字量可以直接从 80C51 的 P0 口送入。

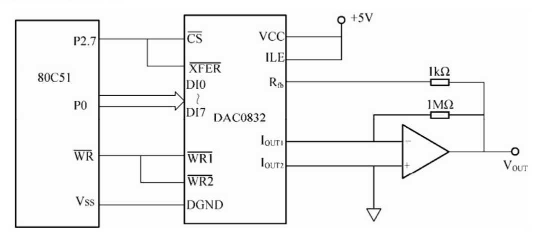


图 6-21　DAC0832 单缓冲方式接口

执行下面几条指令就能完成一次 D/A 转换。

```
MOV     DPTR, #7FFFH        ；指向 DAC0832 口地址(P2.7 为 0)
MOV     A, #data            ；
MOVX    @DPTR, A            ；启动 D/A 转换
```

单极性输出电压 $V_{OUT}=-D \cdot V_{REF}/2^n$，$D$ 为输入数字量，V_{REF} 为基准电压。可见，单极性输出 V_{OUT} 的正负极性由 V_{REF} 的极性确定。当 V_{REF} 的极性为正时，V_{OUT} 为负；当 V_{REF} 的极性为负时，V_{OUT} 为正。

在有些应用场合，还需要双极性模拟输出电压，因此要在编码和电路方面做些改变。图 6-22 为采用偏移二进制码实现 DAC 双极性输出的原理图。

所谓偏移二进制码，就是将 2 的补码的符号位取反，就得到偏移二进制码。由图 6-22 中可见，此时输出 V_{OUT} 是两部分的代数和，一部分是由 V_D 引起的 V_{OUTD}，另一部分是由 V_{REF} 经运放 A2 放大得到的 V_{OUTR}，于是可得

$$\begin{aligned} V_{OUT} &= -(V_{OUTD}+V_{OUTR}) = -2R \cdot V_D/R - 2R \cdot V_{REF}/(2R) \\ &= -2V_D - V_{REF} = 2D \cdot V_{REF}/2^n - V_{REF} = (D/2^{n-1}-1)V_{REF} \\ &= (D-2^{n-1})V_{REF}/2^{n-1} \end{aligned}$$

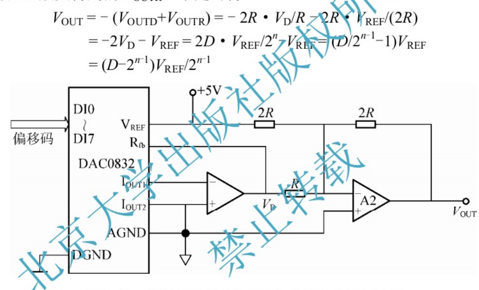


图 6-22　偏移二进制码实现 DAC 双极性输出原理图

将待转换的数字量的偏移二进制码代替上式中的 D，可求出双极性输出 V_{OUT}。若 V_{REF} 由正改为负，那么 V_{OUT} 也反相。例如，数字量 D 的十进制为+127，对应的带符号二进制为 01111111B，偏移二进制码则为 11111111B，此时输出 V_{OUT}(假设 V_{REF} 为正)为

$$V_{OUT} = (255-2^7)V_{REF}/2^7 = (127/128)V_{REF} = V_{REF}-1LSB$$

同理，当数字量 D 的十进制为 −127，对应的带符号二进制为 11111111B，偏移二进制码则为 0000 0001B，此时输出 V_{OUT} 为

$$V_{OUT} = (1-2^7)V_{REF}/2^7 = (-127/128)V_{REF} = -(V_{REF}-1LSB)$$

在双极性输出中，$1LSB=V_{REF}/2^{n-1}=V_{REF}/128$，而单极性输出中 $1LSB=V_{REF}/2^n=V_{REF}/256$。可见，双极性输出时的分辨率比单极性输出时降低 1/2，这是由于对双极性输出而言，最高位作为符号位，只有 7 位数值位。

另外，还可以采用切换基准电压的方法和输出反相的方法来实现双极性输出，限于篇幅这里不做介绍。



2) 双缓冲器工作方式

对于多路 D/A 转换输出，如果要求同步进行，就需要采用双缓冲器同步方式。DAC0832 工作于双缓冲器工作方式时，数字量的输入锁存和 D/A 转换是分两步完成的。首先，CPU 的数据总线分时地向各路 D/A 转换器输入要转换的数字量并锁存在各自的输入锁存器中，然后 CPU 对所有的 D/A 转换器发出控制信号，使各个 D/A 转换器输入锁存器中的数据打入 DAC 寄存器，实现同步转换输出。

图 6-23 为一个二路同步输出的 D/A 转换接口电路。80C51 的 P2.5 和 P2.6 分别选择两路 D/A 转换器的输入锁存器，P2.7 连接到两路 D/A 转换器的 $\overline{XFER}$ 端控制同步转换输出。

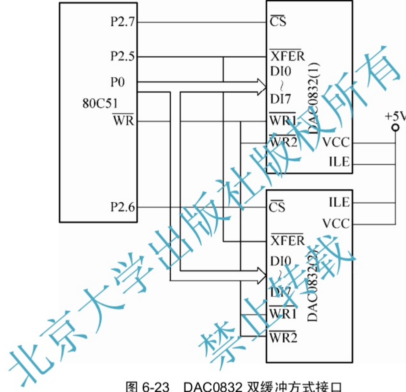
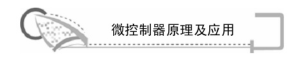


图 6-23　DAC0832 双缓冲方式接口

完成两路 D/A 同步输出的程序如下。

```
MOV     DPTR, #0DFFFH      ; 指向 DAC0832(1) 输入锁存器
MOV     A, #data1
MOVX    @DPTR, A           ; 数字 data1 送入 DAC0832(1) 输入锁存器
MOV     DPTR, #0BFFFH      ; 指向 DAC0832(2) 输入锁存器
MOV     A, #data2
MOVX    @DPTR, A           ; 数字 data2 送入 DAC0832(2) 输入锁存器
MOV     DPTR, #7FFFH       ; 同时启动 DAC0832 (1)、DAC0832(2)
MOVX    @DPTR, A           ; 完成 D/A 转换输出
```

在需要多路 D/A 转换输出的场合，除了采用上述方法外，还可以采用多通道 DAC 芯片。这种 DAC 芯片在同一个封装里有两个以上相同的 D/A 转换器，它们可以各自独立工作。例如，AD7528 是双通道 8 位 DAC 芯片，可以同时输出两路模拟量。



3) 直通工作方式

当 DAC0832 芯片的片选信号 $\overline{CS}$、写信号 $\overline{WR1}$、$\overline{WR2}$ 及传送控制信号 $\overline{XFER}$ 的引脚全部接地，允许输入锁存信号 ILE 引脚接+5V 时，DAC0832 芯片就处于直通工作方式，数字量一旦输入，就直接进入 DAC 寄存器，进行 D/A 转换。

6.4 ADC 接口

将模拟量转换成数字量的器件称为模/数转换器(A/D 转换器，ADC)。

6.4.1 A/D 转换器概述

随着大规模集成电路技术的迅速发展，A/D 转换器新品不断推出。按工作原理分，A/D 转换器的主要种类有：逐次逼近式、双积分式、计数比较式和并行式。

下面介绍最常用的逐次逼近式 A/D 转换器和双积分式 A/D 转换器的转换原理。

1. 逐次逼近式 A/D 转换器的转换原理

图 6-24 是逐次逼近式 A/D 转换器的工作原理图。由图可见，A/D 转换器由比较器、D/A 转换器、逐次逼近寄存器和控制逻辑组成。

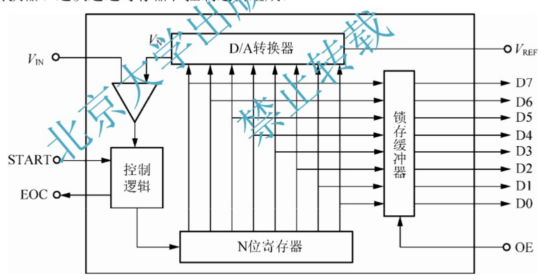


图 6-24 逐次逼近式 ADC 原理图

在时钟脉冲的同步下，控制逻辑先使 N 位寄存器的 D7 位置 1(其余位为 0)，此时该寄存器输出的内容为 80H，此值经 D/A 转换器转换为模拟量输出 V_N，与待转换的模拟输入信号 V_{IN} 相比较，若 V_{IN} 大于等于 V_N，则比较器输出为 1。于是在时钟脉冲的同步下，保留 D7=1，并使下一位 D6=1，所得新值(C0H)再经 D/A 转换器转换得到新的 V_N，再与 V_{IN} 比较，重复前述过程。反之，若使 D7=1 后，经比较，若 V_{IN} 小于 V_N，则使 D7=0，D6=1，所得新值 V_N 再与 V_{IN} 比较，重复前述过程。依此类推，从 D7 到 D0 都比较完毕，转换便

结束。转换结束时，控制逻辑使 EOC 变为高电平，表示 A/D 转换器转换结束，此时的 D7～D0 即为对应于模拟输入信号 V_{IN} 的数字量。

2. 双积分式 A/D 转换器的转换原理

图 6-25 是双积分式 A/D 转换器的工作原理图。控制逻辑先对未知的输入模拟电压 V_{IN} 进行固定时间 T 的积分，然后转为对标准电压进行反向积分，直至积分输出返回起始值。对标准电压的积分时间 T_1(或 T_2)正比于模拟输入电压 V_{IN}。输入电压大，则反向积分时间长。用高频率标准时钟脉冲来测量积分时间 T_1(或 T_2)，即可得到对应于模拟电压 V_{IN} 的数字量。

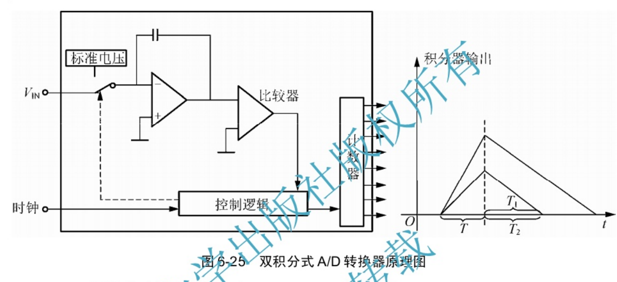


图 6-25　双积分式 A/D 转换器原理图

3. A/D 转换器的主要技术指标

1) 分辨率

A/D 转换器的分辨率是指使输出数字量变化一个相邻数码所需输入模拟电压的变化量。常用二进制的位数表示。例如，12 位 A/D 转换器的分辨率就是 12 位，或者说分辨率为满刻度 FS 的 $1/(2^{12})$。一个 10V 满刻度的 12 位 A/D 转换器能分辨输入电压变化的最小值为 $10V \times 1/(2^{12})=2.4mV$。

2) 量化误差

A/D 转换器把模拟量变为数字量，用数字量近似表示模拟量，这个过程称为量化。量化误差是 A/D 转换器的有限位数对模拟量进行量化而引起的误差。实际上，要准确表示模拟量，A/D 转换器的位数需很大甚至无穷大。一个分辨率有限的 A/D 转换器的阶梯状转换特性曲线与具有无限分辨率的 A/D 转换器转换特性曲线(直线)之间的最大偏差即是量化误差，如图 6-26 所示。

图 6-26(a)中，量化误差为−1LSB。图 6-26(b)中，由于在零刻度处偏移了 1/2LSB，故量化误差为±1/2LSB。ADC 芯片常用偏移的方法减小量化误差。

量化误差和分辨率有关，分辨率高的 ADC 转换器具有较小的量化误差。

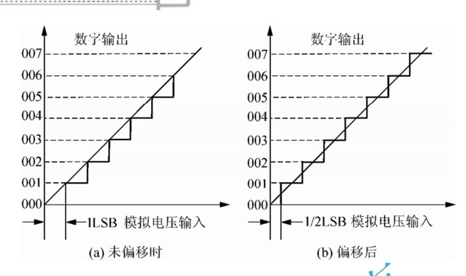


图 6-26 A/D 转换器的转换特性

3) 偏移误差

偏移误差是指输入信号为零时，输出信号不为零的值，所以有时又称为零值误差。假定 A/D 转换器没有非线性误差，则其转换特性曲线各阶梯中点的连线必定是直线，这条直线与横轴相交的点所对应的输入电压值就是偏移误差。

4) 满刻度误差

满刻度误差又称为增益误差。A/D 转换器的满刻度误差是指满刻度输出数码所对应的实际输入电压与理想输入电压之差。

5) 线性度

线性度有时又称为非线性度，它是指转换器实际的转换特性与理想直线的最大偏差。

6) 绝对精度

在一个转换器中，任何数码所对应的实际模拟量输入与理论模拟输入之差的最大值，称为绝对精度。对于 A/D 转换器而言，可以在每一个阶梯的水平中点进行测量，它包括了所有的误差。

7) 转换速率

A/D 转换器的转换速率是能够重复进行数据转换的速度，即每秒转换的次数。而完成一次 A/D 转换器转换所需的时间(包括稳定时间)，则是转换速率的倒数。

6.4.2 微控制器与 ADC0809 的接口设计

ADC0809 是 8 位逐次逼近式、单片 CMOS 集成 A/D 转换器。主要性能如下。

(1) 分辨率为 8 位。

(2) 精度小于±1LSB。

(3) 单+5V 供电，模拟输入电压范围为 0～+5V。

(4) 具有锁存控制的 8 路输入模拟开关。

(5) 可锁存三态输出，输出与 TTL 电平兼容。

(6) 功耗为 15mW。

(7) 不必进行零点和满度调整。





(8) 转换速度取决于芯片外接的时钟频率。时钟频率范围：10～1280kHz。典型值为时钟频率为 640kHz，转换时间约为 100μs。

1. ADC0809 的内部结构及引脚功能

ADC0809 的内部结构如图 6-27 所示。片内带有锁存功能的 8 路模拟多路开关，可对 8 路输入模拟信号分时转换，具有多路开关的地址译码和锁存电路、8 位 A/D 转换器和三态输出锁存器等。

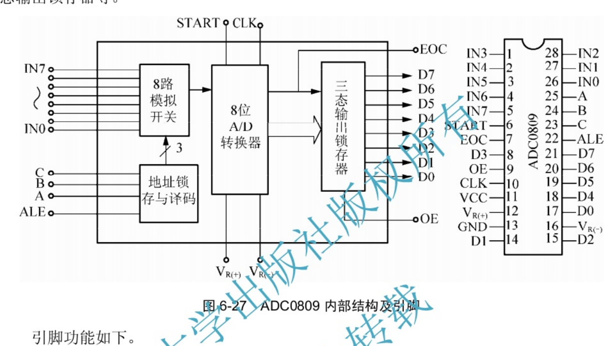


图 6-27　ADC0809 内部结构及引脚

引脚功能如下。

IN0～IN7：8 路模拟量输入端。

D7～D0：8 位数字量输出端。

ALE：地址锁存允许信号输入端。通常向此引脚输入一个正脉冲时，可将 3 位地址选择信号 A、B、C 锁存于地址寄存器内并进行译码，选通相应的模拟输入通道。

START：启动 A/D 转换控制信号输入端。一般向此引脚输入一个正脉冲，上升沿复位内部逐次逼近寄存器，下降沿后开始 A/D 转换。

CLK：时钟信号输入端。

EOC：转换结束信号输出端。A/D 转换期间 EOC 为低电平，A/D 转换结束后 EOC 为高电平。

OE：输出允许控制端，控制输出锁存器的三态门。当 OE 为高电平时，转换结果数据出现在 D7～D0 引脚。当 OE 为低电平时，D7～D0 引脚对外呈高阻状态。

C、B、A：8 路模拟开关的地址选通信号输入端，3 个输入端的信号为 000～111 时，接通 IN0～IN7 对应通道。

$V_{R(+)}$、$V_{R(-)}$：基准电源的正、负输入端。

VCC：电源输入端，+5V。

GND：地。

2. ADC0809 与 MCU 的接口

ADC0809 与 MCU 的接口可以采用查询方式和中断方式。

1) 查询方式

ADC0809 与 MCU 的接口电路如图 6-28 所示。由于 ADC0809 片内无时钟，故利用 80C51 提供的地址锁存允许信号 ALE 经 D 触发器二分频后获得。

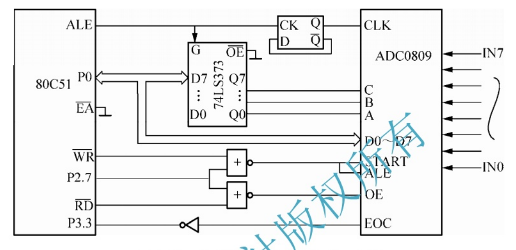


图 6-28 ADC0809 与 MCU 的接口电路

ALE 引脚的频率是 MCU 时钟频率的 1/6，如果 MCU 时钟频率为 6MHz，则 ALE 引脚的频率为 1MHz。再经二分频后为 500kHz，所以 ADC0809 能可靠工作。

由于 ADC0809 具有输出三态锁存器，故其 8 位数据输出线可直接与 MCU 数据总线相连。MCU 的低 8 位地址信号在 ALE 作用下锁存在 74LS373 中，74LS373 输出的低 3 位信号分别加到 ADC0809 的通道选择端 A、B、C 上，作通道编码。MCU 的 P2.7 作为片选信号，与 $\overline{WR}$ 进行或非操作，得到一个脉冲，加到 ADC0809 的 ALE 和 START 引脚上。由于 ALE 和 START 连接在一起，因此 ADC0809 在锁存通道地址的同时也启动转换。在读取转换结果时，用 MCU 的读信号 $\overline{RD}$ 和 P2.7 引脚经或非门后产生的正脉冲作为 OE 信号，用以打开三态输出锁存器。显然，上述操作时，P2.7 应为低电平。ADC0809 的 EOC 端经反相器连接到 MCU 的 P3.3(INT1)引脚，作为查询或中断信号。

下面的程序采用查询方式，分别对 8 路模拟信号轮流取样一次，并依次转换，结果存储到片内 RAM 以 DATA 为起始地址的连续单元中。

```
MAIN:       MOV     R1, #DATA       ；置数据区首地址
            MOV     DPTR, #7FF8H    ；指向 0 通道
            MOV     R7, #08H        ；置通道数
LOOP:       MOVX    @DPTR, A        ；启动 A/D 转换
HERE:       JB      P3.3, HERE      ；查询 A/D 转换结束
            MOVX    A, @DPTR        ；读取 A/D 转换结果
            MOV     @R1, A          ；存储数据
            INC     DPTR            ；指向下一个通道
            INC     R1              ；修改数据区指针
            DJNZ    R7, LOOP        ；8 个通道转换完否？
```



对于上面的程序，也可以采用软件延时的方法读取每次转换的结果，即在启动 A/D 转换器后，延时 100μs 左右，等待转换结果。

2) 中断方式

采用中断方式可大大节省 CPU 的时间。当转换结束时，EOC 向 MCU 发出中断申请信号。响应中断请求后，由中断服务程序读取 A/D 转换结果并存储到片内 RAM 中，然后启动 ADC0809 的下一次转换。

下面的程序采用中断方式，读取 8 个通道的模拟量转换结果。转换结果存储到片内 RAM 以 DATA 为起始地址的连续单元中。



```
            ORG     0013H           ; 外部中断 0 入口地址
            LJMP    INTDATA         ;
            ORG     0030H
MAIN:
            ORG     0100H           ; 数据采集子程序
SAMP:
            SETB    IT1             ; 置边沿触发
            SETB    EX1             ; 允许外部中断 1
            SETB    EA              ; 开中断
            MOV     R1, #DATA       ; 数据缓冲区首址
            MOV     R2, #8          ; 通道计数器
            MOV     DPTR, #7FF8H    ; 指向 IN0 通道
START:
            SETB    F0              ; 置转换结束标志
            MOVX    @DPTR, A        ; 启动 A/D
LOOP:
            JB      F0, LOOP        ; 等待中断
            DJNZ    R2, START       ; 8 路未转换完，则继续
            RET
INTDATA:
            MOVX    A, @DPTR        ; 读转换后的数字量
            MOV     @R1, A          ; 存入片内 RAM
            INC     R1              ; 指向下一个数据存储单元
            INC     DPTR            ; 指向下一个模拟通道
            CLR     F0              ; 清标志 F0，表示一次转换完成
            RETI
```


6.4.3 微控制器与 MAX187 的接口设计

1. MAX187 简介

MAX187 是美国 MAXIM 公司推出的一款 12 位逐次逼近型串行 A/D 转换器，内部含



有大带宽跟踪/保持电路和 4.096V 基准电压，三线串行接口，便于电气隔离，有两种工作模式，转换速率快、功耗低，具有 DIP-8 封装和 SO 封装两种形式。

1) 主要特性

(1) 12 位分辨率。

(2) 单+5V 电源供电。

(3) 兼容 SPI、QSPI、Micro Wire 总线。

(4) 转换时间 5.5～8.5μs，串行输出速率 5MHz 以下。

(5) 功耗低：正常模式下电源电流 1.5～2.5mA，休眠模式下电源电流 2～10μA。

2) 引脚说明

MAX187 的 DIP-8 封装如图 6-29 所示。

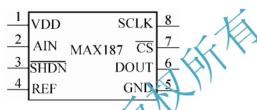


图 6-29　MAX187 的 DIP-8 封装

其引脚功能如下。

(1) VDD：电源端，接+5V 电源。

(2) AIN：模拟量输入，模拟量范围为 0～V_{REF}。

(3) $\overline{\text{SHDN}}$：工作模式控制，高电平为正常工作模式，低电平为休眠模式。

(4) REF：基准电压，接 4.7pF 电容时为内部基准电压 4.096V，外部可接小于 5V 的基准电压。

(5) GND：电源地。

(6) DOUT：数字量串行输出口。

(7) $\overline{\text{CS}}$：使能端，低电平有效。

(8) SCLK：移位脉冲输入，下降沿触发，最高频率为 5MHz。

3) 工作原理

加电 20ms 后基准电压引脚所接电容充电完毕，进入工作状态。当使能端$\overline{\text{CS}}$置为低电平时，内部跟踪/保持器(T/H)进入保持状态并进行转换，转换完毕 DOUT 输出高电平。此时方可在 SCLK 端输入移位脉冲将 12 位转换结果由最高位到最低位依次从 DOUT 端读出，也可以在$\overline{\text{CS}}$端置为低电平 8.5μs 后发送移位脉冲读出转换结果，在读出全部 12 位结果以后将$\overline{\text{CS}}$置高电平。在发送移位脉冲时，由于有一个导前位，所以至少需要 13 个脉冲才能全部移出数据。而且，需要特别注意的是，SCLK 的最高频率是 5 MHz，在编写程序时需注意与系统时钟的匹配问题。

2. MAX187 与 MCU 的接口

图 6-30 为 MAX187 与 80C51 系列 MCU 的接口电路。80C51 的 P1.0 作移位脉冲 SCLK 输出，接 MAX187 的 SCLK；P1.1 作为转换结果输入，接 MAX187 的 DOUT；由于系统

本身扩展外部存储器，故 MAX187 的 $\overline{CS}$ 接 3/8 译码器 Y0 输出端，余下的 P1 口其他引脚留作他用。VDD 电源由稳压芯片 7805 提供；采用内部基准电压，故在 REF 与地之间连接一个 4.7μF 的电容，此时输入模拟量应在 0～4.096V 之间；$\overline{SHDN}$ 接高电平保持工作状态。为保证系统可靠性、减少干扰，采用光电耦合器 4N25 彻底隔离 MCU 与 A/D 转换电路。

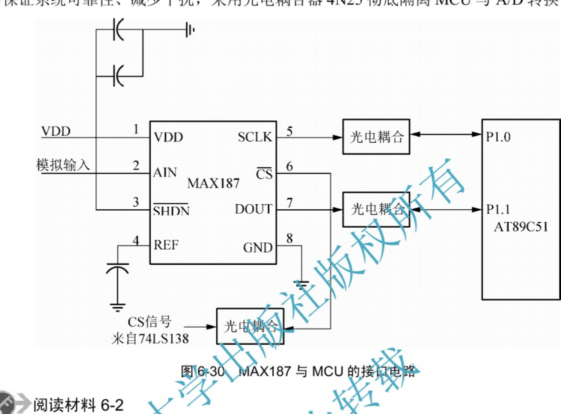


图 6-30　MAX187 与 MCU 的接口电路

阅读材料 6-2

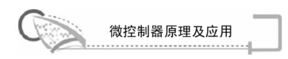
微控制器的应用

现代人类生活中所用的几乎所有电子和机械产品中都会或多或少存在着微控制器(集成单片机)，如手机、计算器、电话、电子玩具、家用电器、掌上电脑及鼠标等电脑配件中都配有 1～2 部单片机。汽车上一般配备 40 多部单片机，而在复杂的工业控制系统上数百台单片机同时工作都有可能。单片机的数量远超过 PC 机和其他机器的总和。现在单片机已经渗透到各个领域，范围大到导弹的导航装置，飞机上仪表的控制，计算机的网络通信与数据传输，工业自动化过程的实时控制和数据处理，小到广泛使用的各种智能 IC 卡，民用豪华轿车的安全保障系统，摄像机、录像机、全自动洗衣机的控制，以及电子宠物、遥控玩具等领域，这些都离不开单片机。因此对单片机的了解和学习是社会发展的必然需求。

本 章 小 结

本章介绍了 MCU 系统中的键盘，两种常用显示器 LED 显示器、LCD 显示器及 ADC、DAC 转换接口。

键盘是 MCU 应用系统中最常用的输入设备，常用的键盘有独立式键盘和矩阵键盘，



当按键数较少时，使用独立式键盘；当按键数较多时，使用矩阵键盘。这两种键盘的硬件都可以直接由 MCU 的 I/O 口构成，软件上则要增加去抖处理程序。

LED 显示器包括 LED 数码管显示器和 LED 点阵显示器两种。LED 数码管显示器所显示的字符较少，形状有些失真，但接口简单，控制方便，适用于规模较小，显示精度要求不高的 MCU 系统中。LED 点阵显示器的字形逼真，可显示的字符较多，但其接口较复杂，控制不够方便，适用于显示精度要求较高的 MCU 系统。

LED 数码管显示器的工作方式有静态显示方式和动态显示方式两种。静态显示方式的显示效果稳定，占用 CPU 时间较少，但需要占用较多的 I/O 口线。因此，在应用中多使用串口扩展串-并转换芯片的方法。动态显示方式可以节约 I/O 口线，但需要编程实现对每位显示器的循环扫描。

LCD 显示器具有功耗低、显示内容丰富等特点。LCD 显示器也可分为段式和点阵式两种。LCD 显示器可以使用 MCU 来直接驱动，但使用专用的 LCD 驱动器和 LCD 显示模块是更实用的两种方法，尤其是 LCD 显示模块，其具有功能强大、易于控制、接口简单等优点，在 MCU 系统中应用较多。

D/A、A/D 转换器是计算机测控系统中常用的芯片，它们可以把数字信号转换成模拟信号输出到外部设备，或把模拟信号转换成数字信号输入计算机。

D/A 转换器主要由基准电压、模拟电子开关、电阻解码网络和运算放大器组成。从分辨率来说，有 8 位、10 位、12 位、16 位之分。位数越多，分辨率越高。

DAC0832 是一种 8 位的 D/A 转换器，输出为电流型。如果需要转换结果为电压，则需外接电流-电压转换电路。DAC0832 有 3 种工作方式，改变 ILE、$\overline{WR1}$、$\overline{WR2}$ 和 $\overline{XFER}$ 的连接方式，可使 DAC0832 工作于单缓冲器、双缓冲器及直通方式。

A/D 转换器的种类有逐次逼近式、双积分式、计数比较式等。逐次逼近式 ADC 由比较器、D/A 转换器、逐次逼近寄存器和控制逻辑组成，ADC0809 即为这种形式的 8 位 8 通道 A/D 转换器。ADC0809 片内带有三态输出缓冲器，其数据输出线可与 MCU 的数据总线直接相连。MCU 读取 A/D 转换结果，可以采用中断方式或查询方式。

MAX187 是一款 12 位逐次逼近型串行 A/D 转换器，内部含有大带宽跟踪/保持电路和 4.096V 基准电压，三线串行接口，便于电气隔离，有正常和休眠两种工作模式，转换速率快、功耗低。

思考题与习题

1. MCU 应用系统中有哪几种键盘类型？

2. 请叙述矩阵键盘的工作原理。中断方式与查询方式的键盘其硬件和软件有何不同？

3. 编制非编码键盘处理程序时，如何去按键抖动？如何判断按键是否释放？

4. 试用 80C51 的 P1 口作 8 个按键的独立式键盘接口，画出其中断方式的接口电路及编制出相应的键盘处理程序。

5. 请叙述 LED 显示器的静态与动态显示原理。什么是 LED 显示器的字符码？



6．LCD 与 LED 显示器在结构和驱动上有何不同？

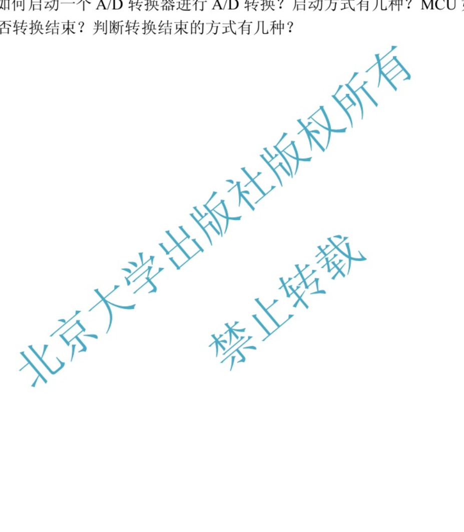
7．试用串行口扩展 4 个 LED 显示器电路，编程使数码管轮流显示 YOUR 和 GOOD，每隔 1s 变换一次。

8. 在什么情况下要使用 D/A 转换器的双缓冲方式？试以 DAC0832 为例绘出双缓冲方式的接口电路。

9．DAC0832 与 80C51 系列 MCU 连接时有哪些控制信号？其作用是什么？

10．为什么 D/A 转换器与 MCU 接口时，必须在 MCU 和 D/A 转换器之间增设锁存器或 I/O 接口芯片？

11．如何启动一个 A/D 转换器进行 A/D 转换？启动方式有几种？MCU 如何判断 A/D 转换器是否转换结束？判断转换结束的方式有几种？



附录 A

Keil μVision4 集成开发环境与 C 语言程序设计

Keil 公司是一家业界领先的微控制器(MCU)软件开发工具的独立供应商。Keil 公司由两家私人公司联合运营，分别是德国慕尼黑的 Keil Elektronik GmbH 和美国德克萨斯的 Keil Software Inc。Keil 公司制造和销售种类广泛的开发工具，包括 ANSI C 编译器、宏汇编程序、调试器、连接器、库管理器、固件和实时操作系统核心(Real-Time Kernel)。有超过 10 万名微控制器开发人员在使用这种得到业界认可的解决方案。其 Keil C51 编译器自 1988 年引入市场以来成为事实上的行业标准，并支持超过 500 种 80C51 变种。Keil 公司在 2007 年被 ARM 公司收购，其两家公司分别更名为 ARM Germany GmbH 和 ARM Inc。

2009 年 2 月，Keil 公司发布了 Keil μVision4，Keil μVision4 引入灵活的窗口管理系统，使开发人员能够使用多台监视器，提供可在虚拟接口上随意放置窗口的完整控制能力。新的用户界面可以更好地利用屏幕空间和更有效地组织多个窗口，提供一个整洁、高效的环境来开发应用程序。新版本支持更多最新的 80C51 兼容芯片及 ARM 芯片，还添加了一些其他新功能，如系统查看器(System Viewer)窗口、多项目工作空间(Multi-Project Workspace)等。

A.1 Keil μVision4 集成开发环境

A.1.1 简介

Keil μVision4 集成开发环境(Integrated Development Environment，IDE)是一个基于 Windows 的开发平台，包含高效的源代码编辑器、项目(Project)管理器和程序生成(MAKE)工具。Keil μVision4 支持所有的 80C51 嵌入式应用工具，它包括 C/C++编译器宏汇编器、连接/定位器和一个 HEX 文件生成器。Keil μVision4 通过以下特性加速 MCU 嵌入式应用系统的开发过程。

(1) 全功能的源代码编辑器。



(2) 器件库用来配置开发工具设置。

(3) 项目管理器用来创建和维护项目。

(4) 集成的 MAKE 工具可以汇编、编译和连接用户的嵌入式应用。

(5) 所有开发工具的设置都是以对话框的形式出现的。

(6) 具有真正的源代码级的对 CPU 和外围器件的调试器。

(7) 高级 GDI 接口用来在目标硬件上进行软件调试及和 Monitor-51 进行通信。

(8) 与开发工具手册、器件数据手册和用户指南有直接的链接。

1. C51 编译器和 A51 汇编器

源代码由μVision4 创建，并被 C51 编译成 A51 汇编。编译器和汇编器从源代码生成可重定位的目标文件。

Keil C51 编译器完全遵照 ANSI C 语言标准，支持 C 语言的所有标准特性。另外，直接支持 80C51 结构的几个特性被添加在里面。

Keil A51 宏汇编器支持 80C51 及其派生系列的全部指令集。

2. LIB51 库管理器

LIB51 库管理器允许从由编译器或汇编器生成的目标文件创建目标库。库是一种被特别地组织过并在以后可以被连接重用的对象模块。当连接器处理一个库时，仅仅那些被使用的目标模块才被真正使用。

3. BL51 连接器/定位器

BL51 连接器/定位器利用从库中提取的目标模块和由编译器或汇编器生成的目标模块创建一个绝对地址的目标模块。一个绝对地址目标模块或文件包含不可重定位的代码和数据。所有的代码和数据被安置在固定的存储器单元中。

此绝对地址目标文件可以用来：写入 EPROM 或其他存储器件；通过μVision4 调试器来模拟和调试；通过仿真器来测试程序。

4. OH51 目标文件转换器

OH51 目标文件转换器可以把前面编译连接好的目标文件转换成能写入 EPROM 中的 HEX 文件。

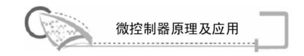
5. μVision4 调试器

μVision4 源代码级调试器是一个理想、快速、可靠的程序调试器。此调试器包含一个高速模拟器，能够模拟整个 80C51 系统，包括片上外围器件和外部硬件。当从器件库中选择器件时，这个器件的特性将自动配置。

μVision4 调试器为在实际目标板上测试程序提供了以下 2 种方法。

(1) 安装 MON51 目标监控器到目标系统并且通过 Monitor-51 接口下载程序。

(2) 利用高级的 GDI(AGDI)接口，把μVision4 调试器绑定到目标系统。



6. Monitor-51

μVision4 调试器支持用 Monitor-51 进行目标板调试。此监控程序驻留在目标板的存储器里，它利用串口和μVision4 调试器进行通信。利用 Monitor-51，μVision4 调试器可以对目标硬件实行源代码级的调试。

7. RTX-51 实时操作系统

RTX-51 实时操作系统是一个针对 80C51 系统的多任务核。RTX-51 实时内核从本质上简化了对实时事件反应速度要求高的复杂应用系统的设计、编程和调试。RTX-51 实时内核是完全集成到 C51 编译器中的，所以使用方便。任务描述表和操作系统的连接由 BL51 连接器/定位器自动控制。

A.1.2 安装

了解了 Keil μVision4 的一些基本概况后，下面开始在计算机上搭建 MCU 的集成开发环境，操作步骤如下。

首先准备 Keil μVision4 安装源文件，双击μVision4 的 Setup 安装文件，弹出 Keil μVision4 安装的欢迎界面，如图 A-1 所示。

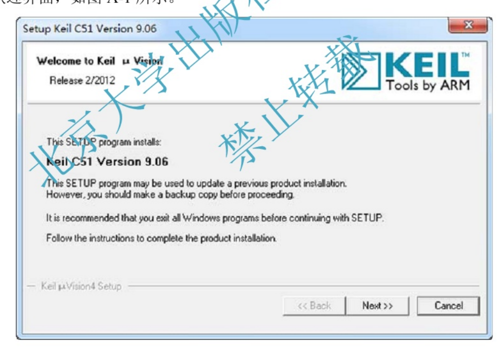


图 A-1　Keil μVision4 欢迎界面

单击“Next”按钮，弹出“License Agreement”对话框，如图 A-2 所示。这里显示了一些用户安装的协议和许可的要求，勾选“I agree to all the terms of the preceding License Agreement”即可，否则无法进入下一步。



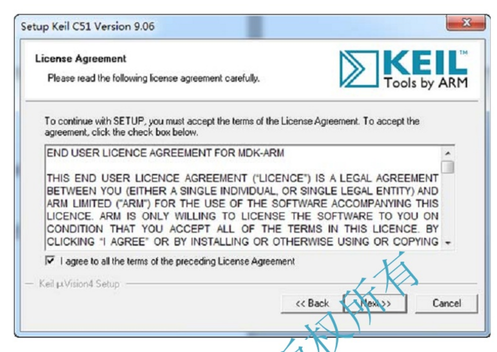


图 A-2　License Agreement 对话框

单击“Next”按钮，弹出“Folder Selection”对话框，如图 A-3 所示。系统默认安装在“C:\Keil”文件夹下。在这里，单击“Browse”按钮，可以选择安装的目录。

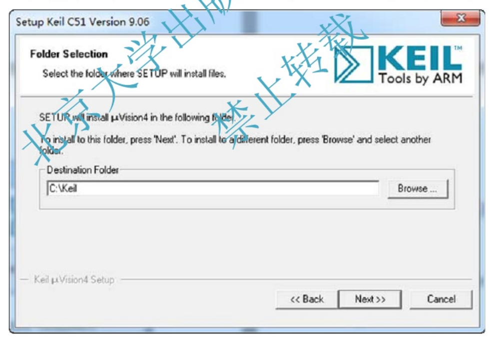


图 A-3　Folder Selection 对话框

单击“Next”按钮，弹出”Customer Information”对话框，如图 A-4 所示。此时，用户需要输入用户名、公司名称和 E-mail，缺一不可。

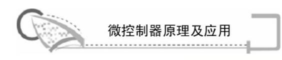




图 A-4 Customer Information 对话框

单击”Next”按钮，下面便开始自动安装。

Keil µVision4 安装完成后，弹出安装完成对话框，如图 A-5 所示。

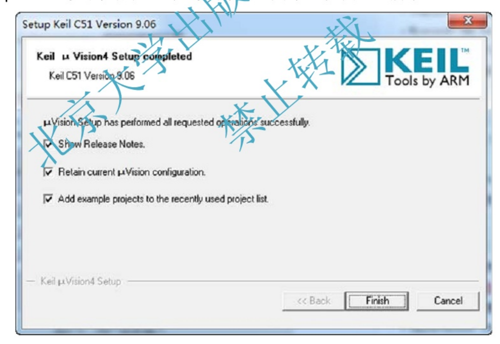


图 A-5 安装完成

这里的几个选项的含义如下。

Show Release Notes：显示安装的版本注释信息。

Retain current µVision configuration：保持当前的设置(如果是第一次安装，则不存在这个选项)。



Add example projects to the recently used project list：添加一些示例程序到当前项目列表中。

最后，单击“Finish”按钮，便可以结束 Keil μVision4 集成开发环境的安装。需要注意的是，刚刚安装完的版本是试用版(Evaluation Version)，代码长度有 2KB 限制。如果代码长度超过 2KB，可与 Keil 公司(www.keil.com)联系，购买 LIC(License ID Code)。

A.1.3　Keil μVision4 集成开发环境界面

安装完成后，会在桌面上出现 Keil μVision4 程序的图标，并在“开始”程序里增加“Keil μVision4”程序项。从“开始”程序里选择“Keil μVision4”程序项或者直接双击桌面上的 Keil μVision4 程序图标，即可启动 Keil μVision4。启动 Keil μVision4 后，如果是第一次运行，则打开工程项目“Hello”，如图 A-6 所示。

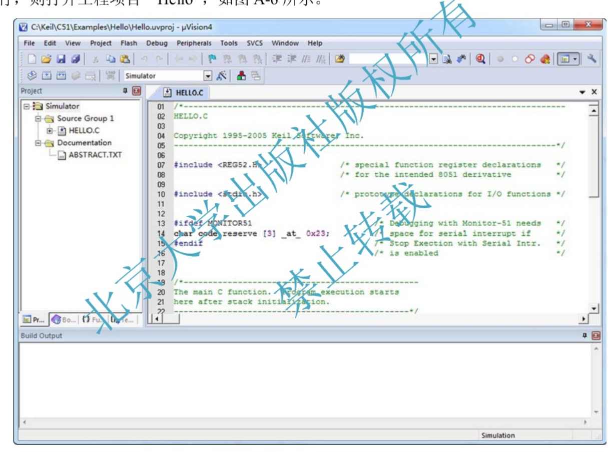


图 A-6　Keil μVision4 集成开发环境界面

这里可以看到，Keil μVision4 集成开发环境具有典型的 Windows 界面风格。整个编程界面主要包括菜单栏、工具栏、项目管理区、源代码工作区和输出信息窗口。另外，还有一些功能窗口将在后面逐步介绍。下面我们将带领读者逐一认识 Keil μVision4 集成开发环境的主要组成部分。

A.1.4　Keil μVision4 菜单命令

Keil μVision4 的菜单栏提供了项目操作、编辑操作、编译调试及帮助等各种常用操作。

所有的操作基本上都可以通过菜单命令来实现。为了快速执行 Keil µVision4 的许多功能，有些菜单命令在工具栏上还有工具条。为了更快速执行一些功能，Keil µVision4 提供了比工具栏上的工具条更为快捷的操作，即快捷键。在 Keil µVision4 集成开发环境中不仅提供了常用功能的默认快捷键，同时用户也可以根据自己的需要自定义快捷键。下面就菜单命令、工具条、快捷键分别进行介绍。

1. File 菜单

File 菜单和标准的 Windows 软件的 File 菜单类似，提供了项目和文件的操作功能。File 菜单各个命令的功能见表 A-1。

表 A-1 File 菜单

菜单命令	工具条	快捷键	功能说明
New		Ctrl+N	创建一个新的空白文件
Open		Ctrl+O	打开一个已存在的文件
Close			关闭当前打开的文件
Save		Ctrl+S	保存当前打开的文件
Save as			当前文件另存为
Save all			保存所有打开的文件
Device Database			打开器件库
License Management			产品注册管理
Print Setup			设置打印机
Print		Ctrl+P	打印当前文件
Print Preview			打印预览
1 .. 10			列出最近打开的源文件或文本文件
Exit			退出 Keil µVision4

2. Edit 菜单

Edit 菜单提供了常用的代码编辑操作命令。Edit 菜单各个命令的功能见表 A-2。

表 A-2 Edit 菜单

菜单命令	工具条	快捷键	功能说明
Undo		Ctrl+Z	取消上次操作
Redo		Ctrl+Y	重复上次操作
Cut		Ctrl+X	剪切选定的内容



续表

菜单命令	工具条	快捷键	功能说明
Copy		Ctrl+C	复制选定的内容
Paste		Ctrl+V	粘贴已复制的内容
Navigate Backwards		Ctrl+Shift+-	光标移动到使用 Find 或 go to line 命令的前一行
Navigate Forwards		Ctr+-	光标移动到使用 Find 或 go to line 命令的后一行
Insert/Remove Bookmark		Ctrl+F2	设置/取消当前行的标签
Go to Next Bookmark		F2	光标移动到下一个标签
Go to Previous Bookmark		Shift+F2	光标移动到上一个标签
Clear All Bookmarks		Ctrl+Shift+F2	清除当前文件的所有标签
Find		Ctrl+F	在当前文件中查找
Replace		Ctrl+H	替换
Find in Files		Ctrl+Shift+F	在多个文件中查找
Incremental Find		Ctrl+I	渐近式寻找
Outlining			源代码概要显示模式
Advanced			各种高级编辑命令
Configuration			颜色、字体等高级配置

3. View 菜单

View 菜单提供了在源代码编辑和仿真调试过程中，各个窗口和工具栏的显示和隐藏命令。View 菜单各个命令的功能见表 A-3。

表 A-3　View 菜单

菜单命令	工具条	功能说明
Status Bar		显示/隐藏状态条
Toolbars		显示/隐藏工具栏
Project Window		显示/隐藏项目管理窗口
Books Window		显示/隐藏参考书窗口
Functions Window	{}	显示/隐藏函数窗口
Templates Window		显示/隐藏模板窗口
Source Browser Window		显示/隐藏资源浏览器窗口


续表

菜单命令	工具条	功能说明
Build Output Window		显示/隐藏输出信息窗口
Find in Files Window		显示/隐藏在所有文件中查找文本窗口
Full Screen		显示/隐藏全屏显示窗口
调试模式下增加的菜单命令		
Command Window		显示/隐藏命令行窗口
Disassembly Window		显示/隐藏反汇编窗口
Symbols Window		显示/隐藏字符变量窗口
Registers Window		显示/隐藏寄存器窗口
Call Stack Window		显示/隐藏堆栈窗口
Watch Windows		显示/隐藏变量子菜单观察窗口
Memory Windows		显示/隐藏存储器子菜单窗口
Serial Windows		显示/隐藏串行口观察子菜单窗口
Analysis Windows		显示/隐藏分析子菜单窗口
Trace		显示/隐藏跟踪子菜单窗口
System Viewer		显示/隐藏外设子菜单窗口
Toolbox Window		显示/隐藏自定义工具条窗口
Periodic Window Update		在程序运行时刷新调试窗口

4. Project 菜单

Project 菜单提供了 MCU 项目的创建、设置和编译等命令。Project 菜单各个命令的功能见表 A-4。

表 A-4　Project 菜单

菜单命令	工具条	快捷键	功能说明
New µVision Project...			创建新项目
New Multi-Project Workspace...			创建多项目工作空间
Open Project...			打开一个已存在的项目
Close Project			关闭当前项目
Export			导出当前一个或多个项目为µVision3 格式
Manage			管理项目的包含文件、库的路径及多项目工作空间



续表

菜单命令	工具条	快捷键	功能说明
Select Device for Target name...			为当前项目选择一个 MCU 类型
Remove object			从当前项目中移除选择的文件或项目组
Options for object		Alt+F7	设置当前文件、项目或项目组的配置选项
Clean target			清除编译过程中创建的中间文件
Build target		F7	编译文件并生成应用文件
Rebuild all target files			重新编译所有文件并生成应用文件
Batch Build...			批量编译文件并生成应用文件
Translate file		Ctrl+F7	编译当前文件
Stop build			停止编译当前项目
1 .. 10			列出最近打开的项目(最多 10 个)

5. Flash 菜单

Flash 菜单提供了下载程序、擦除 MCU 程序存储器等操作。这里的命令需要外部的编程器支持才可以使用。Flash 菜单各个命令的功能见表 A-5。

表 A-5 Flash 菜单

菜单命令	工具条	功能说明
Download		下载 MCU 程序
Erase		擦除程序存储器
Configure Flash Tools...		打开配置工具

6. Debug 菜单

Debug 菜单中的命令大多用于仿真调试过程中，提供了断点、调试方式及逻辑分析等功能。Debug 菜单各个命令的功能见表 A-6。

表 A-6 Debug 菜单

菜单命令	工具条	快捷键	功能说明
Start/Stop Debug Session		Ctrl+F5	开始/停止仿真调试模式
Reset CPU			复位 CPU(MCU)
Run		F5	运行程序，直到遇到一个断点
Stop			停止运行程序


续表

菜单命令	工具条	快捷键	功能说明
Step		F11	单步执行程序，遇到子程序则进入
Step over		F10	单步执行程序，跳过子程序
Step out		Ctrl+F11	程序执行到当前函数的结束
Run to Cursor line		Ctrl+F10	程序执行到光标所在行
Show Next Statement			显示下一条指令
Breakpoints		Ctrl+B	打开断点对话框
Insert/Remove Breakpoint		F9	设置/取消当前行的断点
Enable/Disable Breakpoint		Ctrl+F9	使能/禁止当前行的断点
Disable All Breakpoints			禁用所有断点
Kill All Breakpoints		Ctrl+Shift+F9	取消所有断点
OS Support			打开查看事件、任务及系统信息的子菜单
Execution Profiling			打开一个带有配置选项的子菜单
Memory Map			打开存储器空间配置对话框
Inline Assembly			对某一行进行重新汇编，可以修改汇编代码
Function Editor (Open Ini File)			编辑调试函数和调试配置文件
Debug Settings			设置调试参数

7. Peripherals 菜单

Peripherals 菜单提供了 MCU 各种硬件资源的仿真对话框。这里的所有命令都只在仿真调试环境下才显示并可以使用，而且显示的资源内容随用户选择的 MCU 型号的不同而不同。这里列出一些常用到的 Peripherals 菜单命令的功能见表 A-7。

表 A-7 Peripherals 菜单

菜单命令	功能说明
Interrupt	打开中断仿真对话框
I/O Ports	打开并行端口仿真对话框
Serial	打开串口仿真对话框
Timer	打开定时器仿真对话框
Watchdog	打开看门狗仿真对话框
A/D Converter	打开 A/D 转换器仿真对话框

续表

菜单命令	功能说明
D/A Converter	打开 D/A 转换器仿真对话框
I²C Controller	打开 I²C 总线控制器仿真对话框
CAN Controller	打开 CAN 总线控制器仿真对话框

8. Tools 菜单

Tools 菜单提供了一些第三方软件的支持，如 PC-Lint。用户需要额外安装相应的软件才可以使用。Tools 菜单一般用得较少，这里仅列出各个命令的功能见表 A-8。

表 A-8　Tool 菜单

菜单命令	功能说明
Set-up PC-Lint	配置 PC-Lint 程序
Lint	用 PC-Lint 程序处理当前编辑的文件
Lint All C-Source Files	用 PC-Lint 程序处理项目中所有的 C 源代码文件
Customize Tools Menu...	自定义工具菜单

9. SVSC 菜单

SVSC 菜单提供了程序的版本控制，该菜单下仅包括“Configure Version Control”一个命令，用于配置软件版本。

另外，Windows 菜单下提供了对工作区窗口布局的管理，Help 菜单提供了一些帮助信息，这里不再具体介绍。

A.2　Keil µVision4 汇编语言程序的调试方法

Keil µVision4 集成开发环境中包括一个项目管理器，它可以使基于 80C51 内核的 MCU 应用系统设计变得简单。要创建一个应用，需要按下列步骤操作。

(1) 启动 Keil µVision4，新建一个项目文件并从器件库中选择一个器件。

(2) 新建一个源文件并把它加入到项目中。

(3) 设置目标硬件选项。

(4) 编译项目并生成可以编程到程序存储器的 HEX 文件。

(5) 软件模拟调试及下载到 MCU 中进行仿真调试。

下面通过一个实例，详细介绍如何在 Keil µVision4 集成开发环境中调试 80C51 系列 MCU 的汇编语言程序。

【例 A-1】假设晶振频率为 11.0592MHz，将 MCU 片外 RAM 中 40H～5FH 单元中的内容全部移到片内相同地址区域，并将原数据区全部清零。

A.2.1　启动 Keil μVision4 并创建一个项目

双击桌面 Keil μVision4 程序图标或单击开始菜单中的 Keil μVision4 程序项，启动 Keil μVision4 集成开发环境。

要新建一个项目文件，可以从 Keil μVision4 的 Project 菜单中选择“New Project”项，打开“Create New Project”对话框，如图 A-7 所示。

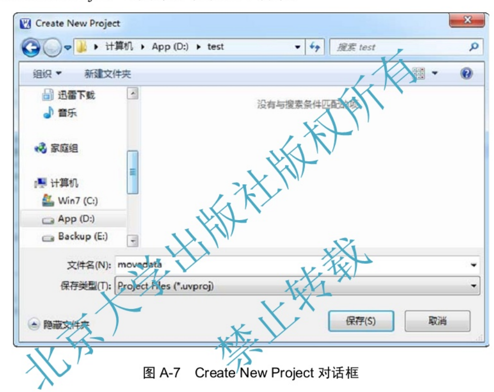


图 A-7　Create New Project 对话框

在此对话框的“文件名”栏中输入项目文件名。建议为每一个项目建立一个独立的文件夹。首先，在下拉列表中选择要保存的位置，最好选择逻辑盘 D 或 E(不要保存在系统盘 C，避免因系统重新安装而丢失文件)。单击“新建文件夹”，得到一个空文件夹，给该文件夹重命名为“test”(文件夹的名字最好能够体现项目名称)。双击该文件夹，在“文件名(N)”栏中输入项目的名称，如“movedata”，创建一个文件名为“movedata.uvproj”的新项目文件。

单击“保存(S)”按钮，将弹出“Select Device for Target‘Target 1’…”对话框，提示为项目选择一个 MCU。在该对话框中，“Data base”列表框中显示出各个 MCU 的生产商。首先找到选用的 MCU 生产商，单击前面的“+”号，显示出 Keil μVision4 所支持的该公司的 MCU 型号列表，单击其中选定的 MCU 型号。本例中，选择 Atmel 公司的型号为 AT89S52 的 MCU，如图 A-8 所示。



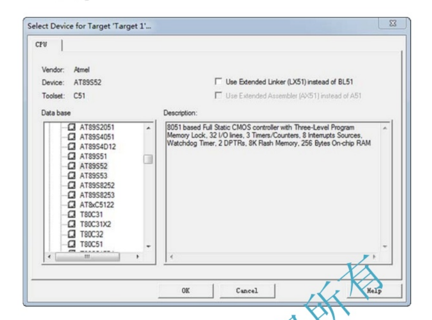


图 A-8　Select Device for Target ‘Target 1’对话框

单击“OK”按钮，弹出如图 A-9 所示的对话框，提示是否将标准 80C51 启动代码复制到项目文件夹中并将该文件添加到项目中去。

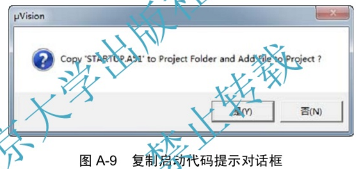


图 A-9　复制启动代码提示对话框

在 Keil µVision4 中，启动代码在复位目标系统后立即被执行。启动代码主要实现以下功能。

(1) 清除内部数据存储器。

(2) 清除外部数据存储器。

(3) 清除外部页存储器。

(4) 初始化 small 模式下的可重入栈和指针。

(5) 初始化 large 模式下的可重入栈和指针。

(6) 初始化 compact 模式下的可重入栈和指针。

(7) 初始化 80C51 硬件栈指针。

(8) 传递初始化全局变量的控制命令或者在没有初始化全局变量时给 main 函数传递命令。

在每一个启动文件中，提供了可供用户自己修改用来控制程序执行的汇编常量。如果只是调试简单程序，可以选择“否(N)”，如果项目复杂可选择“是(Y)”。用户可根据需要

修改启动代码，但一般不建议修改启动代码。

A.2.2 新建一个源文件并把它加入到项目中

从“File”菜单中选择“New”项新建一个源文件，或者单击工具栏上的按钮，打开一个空白的编辑窗口，用户可以输入以下程序源代码。

```
        ORG     0000H
        LJMP    MAIN
        ORG     0040H
MAIN:
        MOV     SP,#70H
        MOV     R0,#40H
        MOV     R1,#20H
LOOP:
        MOV     P2,#00H; 软件仿真时必须有
        MOVX    A,@R0
        MOV     @R0,A
        MOV     A,#00H
        MOVX    @R0,A
        INC     R0
        DJNZ    R1,LOOP
        END
```

从“File”菜单中选择“Save”项或者单击工具栏中的保存按钮，即可保存文件。如果使用汇编语言编写程序，则文件的后缀名是：.asm 或者.a51，如 test.asm，如图 A-10 所示(如果使用 C 语言编写程序，则文件的后缀名是：.c，以下步骤与汇编语言程序相同)。保存后，Keil µVision4 将高亮显示汇编语言语法字符，如图 A-11 所示。

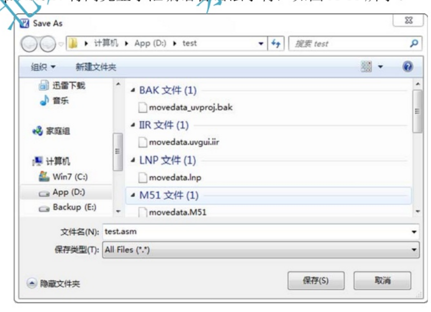


图 A-10 保存源文件



```
test.asm
01           ORG     0000H
02           LJMP    MAIN
03           ORG     0040H
04  MAIN:
05           MOV     SP,#70H
06           MOV     R0,#40H
07           MOV     R1,#20H
08  LOOP:
09           MOV     P2,#00H  ; 软件仿真时必须有
10           MOVX    A,@R0
11           MOV     @R0,A
12           MOV     A,#00H
13           MOVX    @R0,A
14           INC     R0
15           DJNZ    R1,LOOP
16           END
17
```

图 A-11　保存后程序显示界面

源文件创建完成后，就可以将它加入到项目中，如果不加入项目，则无法对此文件操作。Keil µVision4 提供了几种方法让用户把源文件加入到项目中。

(1) 在“Project”(项目管理器)窗口中单击“Target 1”前面的“+”号，展开下一层的“Source Group 1”文件夹，在“Source Group 1”文件夹上单击鼠标右键，弹出快捷菜单，如图 A-12 所示。从弹出的快捷菜单中选择“Add Files to Group ‘Source Group 1’…”项，弹出“Add Files to Group ‘Source Group 1’”对话框，如图 A-13 所示。

在该对话框中，默认的文件类型是“C Source file (*.c)”。若使用汇编语言进行设计，则需要从“文件类型”下拉列表框中选择“Asm Source file (*.s*；*.src；*.a*)”文件类型。这样，以.asm 为扩展名的汇编语言程序文件才会出现在文件列表框中。从文件列表框中选择要加入的文件并双击即可添加到项目中；也可以单击选中文件，然后单击“Add”按钮将该文件加入项目中。添加文件后，对话框不会自动关闭，而是继续等待添加其他文件，用户可单击“Close”按钮，关闭对话框。当给项目添加文件成功后，项目管理器的“Source Group 1”文件夹前面会出现“+”号，单击它可看到 test.asm 文件已经包含在项目中了，双击它即可打开进行修改。

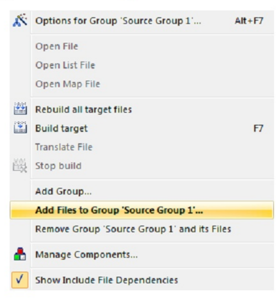


图 A-12　将源文件加入到项目中

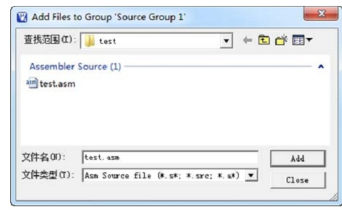


图 A-13　Add Files to Group‘Source Group 1’对话框

A.2.3　设置目标硬件选项

Keil μVision4 允许用户为目标硬件设置选项。可以通过单击工具条图标、菜单“Project”的“Options for Target‘Target 1’…”项或者在“Project Workspace”窗口的“Target 1”上单击鼠标右键，打开“Options for Target‘Target 1’”对话框。在各选项卡中，可以修改目标硬件及所选 MCU 的片上集成器件的所有参数，如图 A-14 所示。

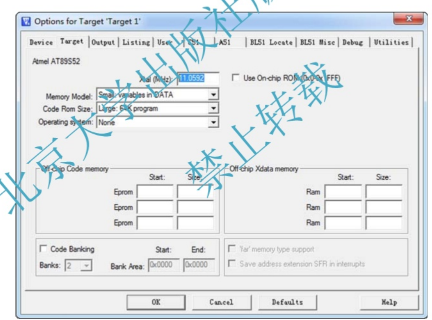


图 A-14　“Options for Target‘Target 1’”对话框

这里主要设置 CPU 的时钟频率、编译器的存储模式等。晶振频率设置应与实际使用的晶振频率相同。如果仅进行软件模拟调试，则采用默认设置即可。

A.2.4　编译项目并生成可以编程到程序存储器的 HEX 文件

单击工具栏中的“Rebuild”图标，可以编译所有的源文件并生成应用。当程序中有语法错误时，Keil μVision4 将在“Build Output”窗口显示错误或者警告信息。双击一行错误提示信息，将打开此信息对应的文件，并定位到语法错误处，如图 A-15 所示。



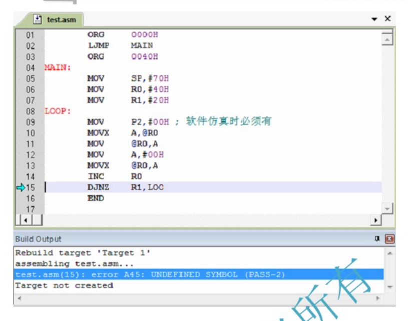


图 A-15　编译出现错误信息时的提示

在错误信息上双击鼠标，光标会自动定位到出现该错误的程序行上。例如，如图 A-15 所示，出现“test.asm(15): error A45: UNDEFINED SYMBOL (PASS-2)”(未定义符号)错误信息，双击该信息，光标定位到出现该错误的行上。用户很容易发现错误，原因是将标号“LOOP”错写成“LOO”，漏掉字母“P”。由输入引起的用户常犯的编译错误还有：错将数字“0”输成字母“o”，使用中文输入法输入了全角逗号(，)和冒号(：)，大于 9FH(如 A8H)的十六进制数忘记在前面加上数字 0(正确写法 0A8H)等。根据错误信息提示，修改程序中出现的错误，直到编译成功为止。一旦编译成功，则显示如图 A-16 所示信息。提示信息最后一行为 “"movedata" - 0 Error(s), 0 Warning(s).”。

需要注意的是，Keil µVision4 默认是不生成 HEX 文件的。这时就需要设置目标硬件选项中的“Output”选项卡，选中“Create HEX File”前面的复选框，如图 A-17 所示。

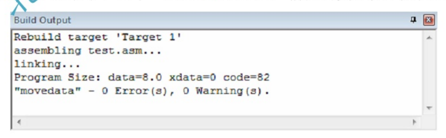


图 A-16　编译成功提示信息

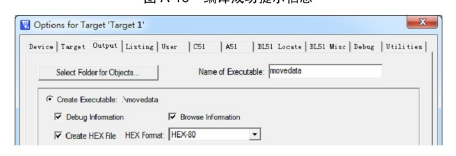


图 A-17　生成 HEX 文件的选项

A.2.5 软件模拟调试及下载到 MCU 中进行仿真调试

一旦编译成功，就可以进行程序的仿真调试了。程序调试有两种方式：一种是软件模拟仿真调试，另一种就是下载到硬件仿真器或者 MCU 中进行在线仿真调试。一般情况下，首先使用软件模拟仿真调试，通过之后，再用硬件仿真器或直接下载到 MCU 中进行在线仿真调试。由于软件模拟仿真调试与在线仿真调试方法基本相同，所以就以软件模拟仿真调试为例，介绍程序的调试方法。

为了对前面编写的程序能够在不连接硬件仿真器或者 MCU 的情况下进行仿真调试(即软件模拟)，需对 Keil μVision4 做一下设置。按照 A.2.3 中设置目标硬件选项时的方法打开“Options for Target ‘Target 1’”对话框，选中“Debug”页，如图 A-18 所示。一般情况下，如果没有进行硬件仿真，则“Use Simulator”单选框是默认选中的，此时，进行软件模拟调试，其他选项不做修改；否则，会选中“Use:”单选框，并在其下拉框中选择相应的硬件驱动，并单击“Settings”按钮对目标仿真硬件进行设置。

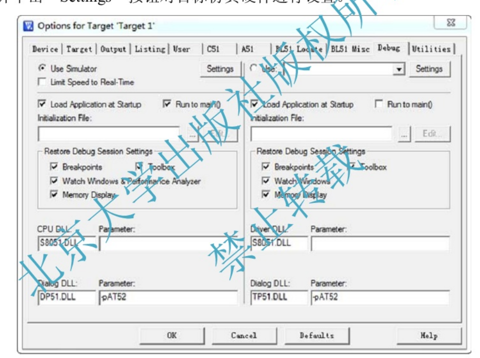


图 A-18 设置“Options for Target ‘Target 1’”对话框中的“Debug”页

通过以上设置，就可以进行软件模拟调试了。

单击工具栏中的“Start/Stop Debug Session”按钮，或者从“Debug”菜单中选中“Start/Stop Debug Session”项(其快捷键为 Ctrl+F5)，开始模拟调试过程。在调试过程中，可以进行如下操作。

1. 连续运行

单击工具栏中的按钮，或者“Debug”菜单中的“Run”(快捷键 F5)，可以使程序全速运行。



2. 停止程序运行

当程序全速运行时，可以单击工具栏中的按钮，或者“Debug”菜单中的“Stop”，使程序停止运行。

3. 复位 CPU

当程序运行过一次以上后，累加器 A、某些寄存器或者其他资源的值修改了，而再次运行需要恢复到初始状态，这时就需要执行复位 CPU 的命令。单击工具栏中的按钮，或者“Debug”菜单中的“Reset CPU”，可以使 MCU 恢复到初始状态。

4. 单步运行

单击工具栏中的按钮，或者“Debug”菜单中的“Step”(快捷键 F11)，可以执行一行程序。如果遇到函数调用，则进入函数内部并单步运行。

5. 单步跳过函数运行

单击工具栏中的按钮，或者“Debug”菜单中的“Step Over”(快捷键 F10)，可以执行一行程序。如果遇到函数调用，则将函数调用看作一行程序运行，不进入函数内部运行。

6. 运行到当前函数结束

这种情况出现在单步运行后进入到函数内部运行程序，通过单击工具栏中的按钮，或者“Debug”菜单中的“Step Out”(快捷键 Ctrl+F11)，以运行到当前函数结束。

7. 运行到光标行

单击工具栏中的按钮，或者“Debug”菜单中的“Run to Cursor Line”(快捷键 Ctrl+F10)，可以执行到光标所在的程序行。

8. 设置断点

在要设置断点的程序行上双击，或者单击工具栏上的按钮，或者“Debug”菜单中的“Insert/Remove Breakpoint”(快捷键 F9)，可以在当前行上插入或者删除断点。只要在当前行上设置了断点，则在当前行的最左边显示一个红色的小方块。连续运行程序后，执行到该行时，程序会暂停运行。此时用户可以查看程序运行的一些中间状态和结果(累加器 A、工作寄存器、SFR、数据存储器等)。

9. 查看寄存器

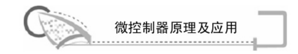
当进入调试状态后，Keil µVision4 集成开发环境中左侧的项目管理器变成寄存器查看器，如图 A-19 所示。用户可以通过这个窗口观察工作寄存器及部分 SFR 的内容。

Registers

Register	Value
Regs	
r0	0x40
r1	0x20
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x70
sp_max	0x70
dptr	0x0000
PC $	C:0x004B
states	10
sec	0.00000364
psw	0x00

Project　Books　Registers

图 A-19　观察寄存器的内容



10. 查看变量及堆栈

在调试状态中，在 Keil μVision4 集成开发环境中的右下侧会出现如图 A-20 所示的窗口，即调用堆栈和变量查看窗口(使用 C 语言编程调试的时候常用)。

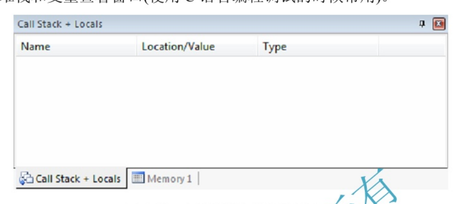


图 A-20　调用堆栈和变量查看窗口

11. 查看存储器

在图 A-20 中单击 Memory1 选项卡则在在 Keil μVision4 集成开发环境中的右下侧会出现如图 A-21 所示的窗口，即存储器查看窗口。

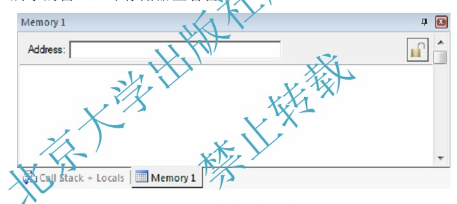


图 A-21　存储器查看窗口

默认情况下，想查看内部 RAM(片内数据存储器)中的内容，需在“Address”编辑框中输入“D:0”并按回车键即可。拖动窗口的左边框可以调整窗口的大小，经过调整，最佳的显示范围如图 A-22 所示。

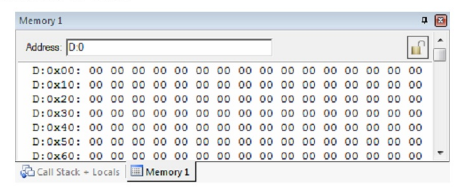


图 A-22　片内数据存储器查看窗口



可以通过“View”菜单中的“Memory Windows”项，添加存储器查看窗口，这样可通过不同的窗口查看不同存储器的内容。例如，可再增加一个窗口查看外部 RAM 中的内容，如图 A-23 所示，在“Address”编辑框中输入“x:0”并按回车键即可。

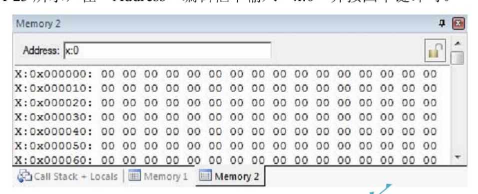


图 A-23　片外数据存储器查看窗口

要改变某个地址单元中的内容，可在上面双击鼠标左键，或者在要修改内容的单元上单击鼠标右键，弹出菜单，选择“Modify Memory at ...”修改。通过弹出菜单，还可修改进制、有符号数、无符号数、ASCII 码等。

“Address”编辑框一般输入格式如下：

×：××××

其中×为：D，查看内部 RAM；X，查看外部 RAM；I，查看间接访问的内部 RAM；C，查看程序 ROM。××××为：查看的起始地址(0000H~FFFFH)。

12. 查看外部设备

单击菜单“Peripherals”可选择查看所选 MCU 集成的不同外部设备。

(1) “Interrupt”打开中断向量表窗口，在窗口里显示了所有的中断向量，如图 A-24 所示。对选定的中断向量可以用窗口下面的复选框进行设置。

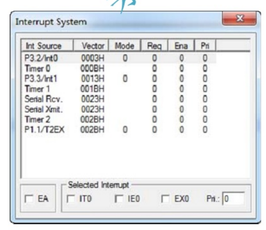
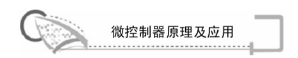


图 A-24　中断向量表窗口

(2) “I/O-Ports”：打开 I/O 端口(P0~P3)的观察窗口，在窗口里显示了程序运行时的端口状态。可以随时查看并修改端口的状态，从而模拟外部的输入。例如，要查看 P2 口的状态，可打开 P2 口的观察窗口，如图 A-25 所示。当运行到第 10 行时，查看窗口如图 A-26



所示。图中标有“√”的复选框表示这一位的值是 1，没有的为 0。对于不同的 MCU，可能图 A-25、图 A-26 的显示略有不同。



图 A-25　刚进入调试状态时 P2 口的查看窗口

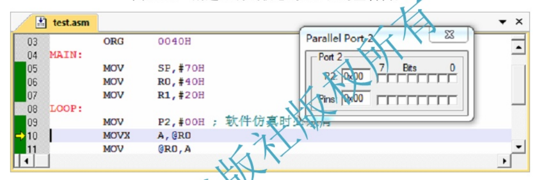


图 A-26　运行到程序行第 10 行时 P2 口的查看窗口

(3) “Serial”：打开串行口的观察窗口，可以随时修改窗口里显示的不同状态。

(4) “Timer”：打开定时器的观察窗口，可以随时修改窗口里显示的不同状态。

除此以外，对于不同公司生产的 MCU，在“Peripherals”菜单中会出现很多与该型号 MCU 相关的外部设备资源菜单项。

掌握了上述的操作过程，就可以进行基本的程序调试工作了。只有不断调试程序，才能逐步积累经验，做到灵活运用，熟练掌握。

A.3　Keil C51 语言

虽然 MCU 上使用的 C 语言都是直接针对具体硬件的，但任何一家公司开发的 C 语言都必须符合 ANSI C 的标准，或者说，要与 ANSI C 兼容。因此，不论哪一家公司为 MCU 开发的 C 语言，其主要部分必然要与 ANSI C 保持一致，不同的只是非 ANSI C 的扩展部分，Keil C51 也不例外。

Keil C51 是一个兼容 ANSI C 的编译器，为了支持 80C51 系列 MCU 加入了一些扩展的内容。C51 编译器与 ANSI C 相比，扩展的内容包括：数据类型、存储器类型、存储模式、指针及函数。其中函数包括定义函数的重入性、指定函数的寄存器组、指定函数的存储模式及定义中断服务程序。

阅读本书的读者请注意，本书没有详细介绍标准的 C 语言，只是介绍 Keil C51 对 ANSI C



的扩展。对于通用的 C 语言部分，如果需要，请查阅相关介绍 C 语言的教材。下面详细介绍 Keil C51 对 ANSI C 的扩展部分。

A.3.1　数据类型

Keil C51 编译器支持的各种规格的数据类型列于表 A-8。除了这些数据类型以外，变量可以组合成结构、联合及数组。

表 A-8　Keil C51 支持的数据类型

数据类型	位数	字节	数值范围
signed char	8	1	−128～+127
unsigned char	8	1	0～255
signed short	16	2	−32 768～+32 767
unsigned short	16	2	0～65 535
signed int	16	2	−32 768～+32 767
unsigned int	16	2	0～65 535
signed long	32	4	−2 147 483 648～+2 147 483 647
unsigned long	32	4	0～4 294 967 295
float	32	4	±1.175 494E−38～±3.402 823E+38
bit	1		0～1
sbit	1		0～1
sfr	8	1	0～255
sfr 16	16	2	0～65 535

表 A-8 所列的数据类型中，关键字 bit、sbit、sfr 和 sfr16 等四种类型在 ANSI C 中是没有的，是 Keil C51 编译器中新增加的。其中，关键字 bit 用于操作 80C51 中的位寻址区，而关键字 sbit、sfr 和 sfr16 用于操作 80C51 的特殊功能寄存器 SFR。

例如，下面的表达式：

```
sfr  P0 = 0x80;        /* 定义 80C51 P0 口的特殊功能寄存器 */
```

声明了一个变量 P0，并且把它和位于 0x80(80C51 的 P0 口)处的特殊功能寄存器联系在一起。

1. bit 类型

bit 数据类型用于定义操作位寻址区的变量，可用于变量声明、参数列表、函数声明和函数返回值等。所有的 bit 变量存放在 80C51 内部存储区的位寻址区。因为这个区域只有 16 字节长，所以最多只能声明 128 个位变量。

一个 bit 变量的声明与其他数据类型相似，例如：

```
static bit gbFlag = 0;          /* 位变量 */
bit bFunc  (                    /* 位函数 */
bit bFlag1,                     /* 位变量 */
bit bFlag2 )                    /* 位变量 */
{
          ⋮
      return(0)                 /* 位返回值 */
}
```

bit 变量的声明中，可包含存储器类型。但是因为 bit 变量存储在 80C51 的内部数据区，只能使用 bdata 存储类型，不能使用别的存储类型。例如：

```
int bdata iBase;             /   * 在直接访问数据区定义一个整型变量 iBase */
char bdata cAry[4];             /* 在间接访问数据区定义一个数组 cAry */
bit mybit0 = iBase ^ 0;
bit mybit15 = iBase ^ 15;
bit bAry07 = cAry[0] ^ 7;
bit bAry37 = cAry[3] ^ 7;
```

bit 变量和 bit 声明有以下限制：

(1) 如果在函数中禁止使用中断(#pragma disable)或者函数中包含有明确的寄存器组切换(using n)，则该函数不能返回一个位值。否则，在编译时会产生编译错误。

(2) 一个位不能被声明为一个指针，如“bit *bPtr;”是错误的。

(3) 不能声明使用一个 bit 类型的数组，如“bit bArr[5];”是错误的。

2. sfr 类型

sfr 和 C 语言其他类型的变量声明是一样的。例如：

```
sfr P0 = 0x80;         /* P0 口，地址为 80H */
sfr P1 = 0x90;         /* P1 口，地址为 90H*/
sfr P2 = 0xA0;         /* P2 口，地址为 0A0H */
sfr P3 = 0xB0;         /* P3 口，地址为 0B0H*/
```

P0、P1、P2 和 P3 是声明的 SFR 名。在等号(=)后指定的地址必须是一个常数值，不允许用带操作数的表达式。标准的 80C51 系列支持 SFR 地址从 0x80 到 0xFF。

3. sfr16

Keil C51 编译器提供的 sfr16 数据类型，可以将两个 8 位的 SFR 作为一个 16 位的 SFR 来访问。访问该 16 位的 SFR 只能是低字节跟着高字节，即将低字节的地址用作 sfr16 声明的地址。例如：



```
sfr16 T2 = 0xCC;            /*定义 Timer2 的 16 位数据寄存器， TL2 的地址为 0CCH,TH2
                              的地址为 0CDH */
```

在这个例子中，定时器 T2 的 16 位的数据寄存器被声明为 16 位 SFR。当然，这个 16 位的数据寄存器可以声明为 2 个 8 位的数据寄存器。例如：

```
sfr TL2 = 0xCC;             /*定义 Timer2 的 16 位数据寄存器的低 8 位， TL2 的地址为
                              0CCH */
sfr TH2 = 0xCD;             /*定义 Timer2 的 16 位数据寄存器的高 8 位， TH2 的地址为
                              0CDH */
```

sfr16 声明和 sfr 声明遵循相同的原则：任何符号名可用在 sfr16 的声明中。等号(=)指定的地址必须是一个常数值，不允许使用带操作数的表达式，而且必须使用 SFR 的低位和高位字节中的低位字节的地址。

4. sbit 类型

在 80C51 系列 MCU 中，经常需要访问 SFR 中的某些位，这时需使用关键字 sbit，利用它可以定义可位寻址的对象。定义方法有如下三种。

1) sbit 位变量名=位地址

这种方法将位的绝对地址赋给位变量，位地址必须位于 0x80～0xFF 之间。例如：

```
sbit OV = 0xD2;
sbit CY = 0xD7;
```

2) sbit 位变量名 = SFR 名 ^ 位位置

当可位寻址的位位于 SFR 中的时候，可采用此方法。“位位置”是一个 0~7 之间的常数。例如：

```
sfr PSW = 0xD0;
sbit OV = PSW ^ 2;
sbit CY = PSW ^ 7;
```

3) sbit 位变量名 = 字节地址 ^ 位位置

这种方法以字节地址作为基地址，该字节地址必须位于 0x80～0xFF 之间。“位位置”是一个 0～7 之间的常数。例如：

```
sbit OV = 0xD0 ^ 2;
sbit CY = 0xD0 ^ 7;
```

A.3.2　存储器类型

80C51 的存储区域有两个特点：①程序存储器和数据存储器是截然分开的；②特殊功能寄存器与内部数据存储器是统一编址的。

C51 编译器支持 80C51 的这种存储器结构，能够访问 80C51 的所有存储器空间。针对 80C51 存储空间的多样性，提出了修饰存储空间的修饰符，用以指明所定义的变量应分配

在什么样的存储空间，见表 A-9。

表 A-9　存储空间类型说明符

存储器类型	描　　述
code	程序空间(64KB)；通过 MOVC　@A+DPTR 访问
data	直接访问的内部数据存储器；访问速度最快(128B)
idata	间接访问的内部数据存储器；可以访问所有的内部存储器空间(256B)。
bdata	可位寻址的内部数据存储器；可用字节方式也可用位方式访问(16B)。
xdata	外部数据存储器(64KB)；通过 MOVX　@DPTR 访问
pdata	分页的外部数据存储器(256 字节)；通过 MOVX　@Ri 访问

1. 程序存储区

程序的代码(CODE)存储区是只读的，不能写入。硬件决定最多可能有 64KB 的程序存储区。用 code 标识符来访问片内、片外统一编址的程序存储区，寻址范围为 0000H～FFFFH。在此空间存放程序代码、数据和表格。用间接寻址的方式访问程序存储区数据，如“MOVC　A,@A+DPTR”或“MOVC　A,@A+PC”。

2. 内部数据存储区

内部的数据存储区是可读、可写的。80C51 系列最多可有 256B 的内部数据存储区。内部数据区可以分成三个不同的存储类型 data、idata 和 bdata。

data 存储类型标识符通常是指低 128B 的内部数据区，为片内直接寻址的 RAM 空间，寻址范围为 0～127。在此空间内存取速度最快。

idata 存储类型标识符是指全部 256 个字节的内部存储区，为片内间接寻址的 RAM 空间，寻址范围为 0～255。寻址方式为“MOV　@Ri”。由于只能间接寻址，访问速度比直接寻址慢。

bdata 存储类型标识符是指可位寻址的 16B 内部存储区(20H～2FH)，位地址范围为 0～127。

本空间允许按字节和按位寻址。在本区域可以声明可位寻址的数据类型。

3. 外部数据存储区

外部数据存储区是可读、可写的。可通过一个数据指针加载一个地址来间接访问外部数据区。因此，访问外部数据存储区比访问内部数据存储区来得慢。

外部数据存储区最多可有 64KB，这些地址不一定都用来作为数据存储区，因为硬件设计可能把外围设备影射到该存储区。

编译器提供两种不同的存储类型来访问外部数据——xdata 和 pdata。

xdata 存储类型标识符是指外部数据存储区(64KB)内的任何地址，寻址范围为 0000H～FFFFH。寻址方式为“MOVX　@DPTR”。

pdata 存储类型标识符仅指一页或 256B 的外部数据存储区，寻址范围为 00H～FFH。



寻址方式为“MOVX @Ri”。

在定义变量时，通过指明存储器类型可以将所定义的变量存储在指定的存储区域中。访问内部数据存储器将比访问外部数据存储器快得多，因此应该把频繁使用的变量放置在内部数据存储器中，把很少使用的变量放在外部数据存储器中。

在变量的声明中，可以包括存储器类型和 signed 或 unsigned 属性。例如：

```
char data var1;
char code text[ ] = "ENTER PARAMETER";
unsigned long xdata array[100];
float idata x,y,z;
unsigned int pdata dimension;
unsigned char xdata vector[10][4][4];
char bdata flags;
```

如果在变量的定义中没有包括存储器类型，将自动选用默认的存储器类型。

A.3.3 存储器模式

如果省略存储器类型，则系统会按编译模式 SMALL、COMPACT 或 LARGE 所规定的默认存储器类型去指定变量的存储区域。无论什么存储模式都可以在任何的 80C51 存储区范围声明变量，然而把最常用的命令如循环计数器和队列索引放在内部数据区可以显著提高系统性能。需要指出的是，变量的存储种类与存储器类型是完全无关的。

1. SMALL 模式

SMALL 存储模式把所有函数变量和局部数据段放在 80C51 系统的内部数据存储区，这样访问数据非常快，但 SMALL 存储模式的地址空间受限。在写小型的应用程序时，变量和数据放在 data 内部数据存储器中是很好的，因为访问速度快，但在较大的应用程序中 data 区最好只存放小的变量、数据或常用的变量(如循环计数、数据索引)，而大的数据则放置在别的存储区域。

2. COMPACT 模式

COMPACT 存储模式中所有的函数、程序变量和局部数据段定位在 80C51 系统的外部数据存储区。外部数据存储区可有最多 256B(一页)，在本模式中外部数据存储区的短地址用 Ri。

3. LARGE

LARGE 存储模式中所有函数、过程的变量和局部数据段都定位在 80C51 系统的外部数据区。外部数据区最多可有 64KB，这要求用 DPTR 数据指针访问数据。

一般情况下，应该使用小(SMALL)模式，它产生最快、最紧凑、效率最高的代码。在定义变量时，最好指定存储器类型。只有当应用不可能在 SMALL 模式下操作时，才需要往上增加存储模式。

A.3.4 指针

C51 编译器支持用星号(*)进行指针声明，可以用指针完成标准 C 语言中的所有操作。由于 80C51 及其派生系列所具有的独特结构，C51 编译器支持两种不同类型的指针：通用指针和存储器指针。

1. 通用指针

通用或未定型的指针的声明和标准 C 语言中一样。如：

```
char   *s;                    /* 字符指针 */
int   *numptr;                /* 整型指针 */
long  *state;                 /* 长整型指针 */
```

通用指针需要 3 个字节来存储。第一个字节用来表示存储器类型，第二个字节是指针的高字节，第三个字节是指针的低字节。

通用指针可以用来访问所有类型的变量，而不管变量存储在哪个存储空间中，因而许多库函数都使用通用指针。通用指针很方便，但是也很慢。在所指向目标存储空间不明确的情况下，它们用得最多。

2. 存储器指针

存储器指针或类型确定的指针在定义时要包含一个存储器类型说明，并且总是指向此说明的特定存储器空间。例如：

```
char data  *str;          /* 指向 data 区域的字符串 */
int xdata  *numtab;       /* 指向 xdata 区域的 int */
long code  *powtab;       /* 指向 code 区域的 long */
```

正是由于存储器类型在编译时已经确定，通用指针中用来表示存储器类型的字节就不再需要了。指向 idata、data、bdata 和 pdata 的存储器指针使用一个字节来保存；指向 code 和 xdata 的存储器指针用两个字节来保存。

由此可见，使用存储器指针比通用指针效率要高，速度要快，但存储器指针的使用不是很方便，故只有在所指向目标存储空间明确并不会变化的情况下才用它。

A.3.5 函数

1. 重入函数

函数的嵌套调用是指当一个函数正被调用尚未返回时，又被本函数或其他函数再次调用的情况，只有等到后次调用返回到了本次，本次被暂时搁置的程序才能恢复接续原来的正常运行，直到本次返回。允许被嵌套调用的函数必须是可重入函数，即函数应具有可重入性。

通常情况下，C51 函数一般是不能被递归调用的，这是由于函数参数和局部变量是存储在固定的地址单元中的。重入函数需要使用重入堆栈，这种堆栈是在存储模式所指的空间内从顶端另行分配的一个非覆盖性的堆栈。该堆栈将被嵌套调用的每层参数及局部变量



一直保留到由深层返回到本层，而又终止本层的返回。

在一个基本函数的基础上添加 reentrant 说明，从而使它具有重入特性。例如：

```
int  calc (char i, int b)  reentrant
{
        int x;
        x = table [i];
        return (x * b);
}
```

在实时应用中以及中断服务程序代码和非中断程序代码必须共用一个函数的场合中，经常用到重入函数。

需要注意的是，不应将全部程序声明为重入函数，因为会增加目标代码的长度并减慢运行速度，故应该选择那些必需的函数作为重入函数。

2. 函数使用指定的寄存器组 using n

函数使用指定寄存器组的定义说明如下：

void 函数标识符(形参表)using n

其中 n=0～3，为寄存器组号，对应 80C51 中的 4 个寄存器组。函数使用了 using n 后，C51 编译器自动在函数的汇编码中加入如下的函数头段和尾段：

```
{
        push      psw
        mov       psw, #与寄存器组号 n 有关的常量
        ⋮
        pop       psw
}
```


应该注意的是，using n 不能用于有返回值的函数，因为 C51 的返回值是放在寄存器中的，而返回前寄存器组却改变了，这将导致返回值发生错误。

3. 函数使用指定的存储模式

80C51 存储空间多样，修饰存储空间的修饰符用以指明所定义的变量应分配在什么样的存储空间，其定义格式为

类型说明符 函数标识符(形参表) 存储模式修饰符{small，compact，large}

其中，修饰符可用 small、compact、large 三者中的一个。

存储模式为本函数的参数和局部变量指定的存储空间，在指定了存储模式之后，该空间将再也不随编译模式而变。例如：

```
extern int func (int i, int j) large;          /* 修饰为大模式 */
```

4. 中断服务程序

C51 编译器允许用 C 语言创建中断服务程序，只需关心中断号和寄存器组的选择，编

译器自动产生中断向量和程序的入栈及出栈代码。

在函数声明时 interrupt m 将把所声明的函数定义为一个中断服务程序，其格式为

void 函数标识符(void) interrupt m

其中，m=0～31。0 对应于外部中断 0；1 对应于定时器 0 中断；2 对应于外部中断 1；3 对应于定时器 1 中断；4 对应于串行口中断；其他为预留。

从定义中可以看出，中断的函数必须是无参数、无返回值的函数。例如：

```
unsigned int interruptcnt;
unsigned char second;
void timer0 (void) interrupt 1 using 2
{
if (++interruptcnt == 4000)          /* 计数到 4000 */
{
second++;                            /* 秒计数器 */
interruptcnt = 0;                    /* 清除中断计数器 */
}
}
```

A.4 C51 语言与汇编语言的混合编程

C51 语言编程与汇编语言编程各有所长。使用 C51 语言，开发速度快，可读性、可维护性、可移植性好；而使用汇编语言，则可以更为充分地利用芯片的软、硬件资源，使程序代码的执行效率高。为了发挥 C51 语言与汇编语言两种语言各自的优势，可混合编程。这一点特别适用于要求占用空间小、有严格时间限制的子程序设计，这类子程序总是希望用汇编语言来编写，然后由 C51 语言主程序来调用；或者直接在 C51 语言中嵌入汇编语言。

A.4.1 Keil C51 调用汇编函数

通过一个例子，介绍在 C51 程序中调用汇编函数的方法。在这个例子里，外部函数的入口参数是一个字符型变量和一个位变量，返回值是一个整型变量。例中，先用 C51 写出这个函数的主体，然后用 SRC 控制指令编译产生.asm 文件，进一步修改这个.asm 文件就得到我们所要的汇编函数。该方法让编译器自动完成各种段的安排，提高了汇编程序的编写效率。

1. 建立项目

按普通 C51 程序方法建立项目，在里面导入 main.c 文件和 cfunc.c 文件。

```
//main.c 文件
#include < reg51.h >
#define uchar unsigned char
#define uint unsigned int
extern uint AFUNC(uchar v_achr,bit v_bflag);
void main()
```



```
{
    bit BFLAG;
    uchar mav_chr;
    uint mvintrslt;
    mav_chr = 0xd4;
    BFLAG = 1;
    mvintrslt = AFUNC(mav_chr , BFLAG);
}
//CFUNC.c 文件
#define uchar unsigned char
#define uint unsigned int
uint AFUNC(uchar v_achr , bit v_bflag)
{
    uchar tmp_vchr;
    uint  tp_vint;
    tmp_vchr = v_achr;
    tp_vint = (uint)v_bflag;
    return tmp_vchr + (tp_vint << 8);
}
```

2. 设置文件选项

在项目管理器窗口中，在将要得到汇编代码的 C 文件“cfunc.c”上单击鼠标右键，弹出菜单，选择“Options for File‘cfunc.c’”，单击右边的“Generate Assembler SRC File”和“Assemble SRC File”，使复选框由灰色变成黑色(有效)状态，如图 A-27 所示。在选择时需注意，该选项有 3 种状态：未选中、无效(灰色)和有效(黑色)。

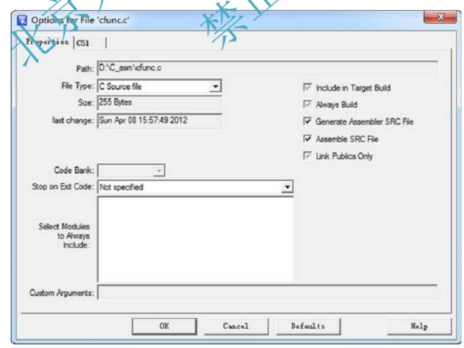


图 A-27　Options for File‘cfunc.c’选项



3. 向项目中添加库文件

根据选择的编译模式，把相应的库文件(如 Small 模式时，是 Keil\C51\Lib\C51S.Lib)加入工程中，该文件必须作为项目的最后文件。

4. 生成汇编语言文件并调整项目文件

编译这个项目后将会产生一个 CFUNC.SRC 的文件，将这个文件改名为 CFUNC.A51 (也可以通过编译选项直接产生 CFUNC.A51 文件)，然后在项目里去掉库文件(如 C51S.Lib)和 CFUNC.c,而将 CFUNC.A51 添加到项目里。

```
; .\cfunc.SRC generated from: cfunc.c
; COMPILER INVOKED BY:
;           C:\Keil\C51\BIN\C51.EXE cfunc.c BROWSE DEBUG OBJECTEXTEND
SRC(.\cfunc.SRC)


NAME    CFUNC

?PR?_AFUNC?CFUNC      SEGMENT CODE
?BI?_AFUNC?CFUNC      SEGMENT BIT OVERLAYABLE
    PUBLIC  ?_AFUNC?BIT
    PUBLIC  _AFUNC

    RSEG  ?BI?_AFUNC?CFUNC
?_AFUNC?BIT:
    v_bflag?041:   DBIT   1
; //CFUNC.c 文件
;
; #define uchar unsigned char
; #define uint unsigned int
;
; uint AFUNC(uchar v_achr , bit v_bflag)

    RSEG  ?PR?_AFUNC?CFUNC
_AFUNC:
    USING   0
            ; SOURCE LINE # 6
;---- Variable 'v_achr?040' assigned to Register 'R7' ----
; {
            ; SOURCE LINE # 7
;    uchar tmp_vchr;
;    uint  tp_vint;
;
```



```
;     tmp_vchr = v_achr;
          ; SOURCE LINE # 11
;---- Variable 'tmp_vchr?042' assigned to Register 'R5' ----
    MOV      R5,AR7
;     tp_vint = (uint)v_bflag;
          ; SOURCE LINE # 12
    MOV      C,v_bflag?041
    CLR      A
    RLC      A
;---- Variable 'tp_vint?043' assigned to Register 'R6/R7' ----
;     return tmp_vchr + (tp_vint << 8);
          ; SOURCE LINE # 13
    MOV      R6,A
    MOV      R4,#00H
    CLR      A
    ADD      A,R5
    MOV      R7,A
    MOV      A,R4
    ADDC     A,R6
    MOV      R6,A
; }          ; SOURCE LINE # 14
?C0001:
    RET
; END OF _AFUNC

    END
```


5. 编译项目

再次编译这个项目，到此已经得到汇编函数的主体，修改函数里面的汇编代码就得到所需的汇编函数了。

A.4.2　在 Keil C51 中直接嵌入汇编语言

1. C51 中嵌入汇编语言程序的格式

要在 C51 文件中嵌入汇编语言程序，需要按照如下格式加入：

#pragma asm
汇编语言程序
#pragma endasm

在上例中，将 main.c 文件内容修改成如下所示。

```
//main.c 文件
#include "reg52.h"
```

```
#define uchar unsigned char
#define uint unsigned int
extern uint AFUNC(uchar v_achr,bit v_bflag);
void main()
{
    bit     BFLAG;
    uchar   mav_chr;
    uint    mvintrslt;
    mav_chr = 0xd4;
    BFLAG = 1;
    mvintrslt = AFUNC(mav_chr , BFLAG);
    #pragma asm
    MOV     P1,mvintrslt?042
    MOV     P2,mvintrslt?042+01H
    #pragma endasm
}
```

即将调用函数 AFUNC()得到的返回值通过 P1 口、P2 口输出。

2. 设置文件选项

在项目管理器窗口中，在将要得到汇编代码的 C 文件“main.c”上单击鼠标右键，弹出菜单，选择“Options for File ‘main.c’”，单击右边的“Generate Assembler SRC File”和“Assemble SRC File”，使复选框由灰色变成黑色(有效)状态。

3. 向项目中添加库文件

根据选择的编译模式，把相应的库文件(如 Small 模式时，是 Keil\C51\Lib\C51S.Lib)加入工程中，该文件必须作为项目的最后文件。

4. 编译并生成目标代码

编译整个项目即可得到用户需要的目标代码。

使用此方法可以在 C51 源代码的任意位置嵌入汇编语言程序。但是，需要注意的是，在直接使用形参时，不同的优化级别下产生的汇编代码可能有所不同。

A.5 C 语言程序举例

例 A.1(汇编语言见例 4.1)

```
#include <reg52.h>
#include <intrins.h>
sbit key = P1^7;
sbit led = P1^0;
void delay(unsigned int time);
```



```
void main(void)
{
        P1 = 0x80;              //P1.7 写“1”，作为输入口线
        while(1)
        {
            while(key);         //检测 P1.7 是否为 0，是，则按键按下
            delay(12500);       //延时，去除按键抖动
            while(key);         //检测 P1.7 是否为 0，是，则确认按键按下
            while(!key);        //检测按键是否抬起
            led = !led;         //LED 点亮或熄灭
        }
}

void delay(unsigned int time)
{
        while(time--)
        {
            _nop_();
        }
}
```

例 A.2(汇编语言见例 4.2)

```
#include<reg52.h>
sbit square = P1^0;
void main()
{
        TMOD = 0x10;            //T1 为方式 1
        TH1 = 0xFE;             //设置计数初值
        TL1 = 0X0C;
        EA = 1;                 //允许中断
        ET1 = 1;                //允许 T1 中断
        TR1 = 1;                //启动 T1
        while(1);               //等待中断
}
void timer1(void) interrupt 3
{
        TH1 = 0xFE;             //重新设置初值
        TL1 = 0x0C;
        square = !square;       //定时 1ms 时间到，输出取反
}
```

例 A.3(汇编语言见例 4.3)

```
#include<reg52.h>
sbit square = P1^0;
void main()
{
        TMOD = 0x02;            //T0 为方式 2
        TH0 = 0x9C;             //设置计数初值
        TL0 = 0x9C;
        EA = 1;                 //允许中断
        ET0 = 1;                //允许 T0 中断
        TR0 = 1;                //启动 T0
        while(1);               //等待中断
}
void timer0(void) interrupt 1
{
        square = !square;       //定时 200μs 时间到，输出取反
}
```

例 A.4(汇编语言见例 4.4)

```
#include<reg52.h>
sbit ex_int1 = P3^3;            //外部中断 1
void display(int value);
void main()
{
        int clock_num;          //机器周期个数
        TMOD = 0x90;            //T0 为方式 1，GATE=1
        TH1 = 0x00;             //设置计数初值
        TL1 = 0x00;
        while(ex_int1);         //等待外部中断 1 降低
        TR1 = 1;                //如果外部中断 1 为低，启动 T1，计数器开始计数
        while(!ex_int1);        //等待外部中断 1 升高
        while(ex_int1);         //外部中断 1 为高，等待外部中断 1 降低
        TR1 = 0;                //T1 停止计数
        clock_num = TL1;        //T1 计数值赋给变量 clock_num
        display(clock_num);     //将计数值送到显示器显示
        while(1);
}
void display(int value)
{
        //以机器周期个数的形式显示正脉冲宽度
}
```



例 A.5(汇编语言见例 4.5)

```
#include<reg52.h>
unsigned long distance _at_ 0x30;
void main(void)
{
        IT0 = 0;                //设置下降沿触发方式
        PX0 = 1;                //置外部中断 0 高优先级
        EX0 = 1;                //允许外部中断 0
        EA = 1;                 //开 CPU 中断
        distance = 0;           //初始化里程计数器
        while(1);               //等待中断
}
void ex_int0() interrupt 0
{
        distance += 2;          //里程计数器+2
}
```

例 A.6(汇编语言见例 4.6)

```
#include <reg52.h>
#define uchar unsigned char
uchar code table[]={0x03,0x9F,0x25,0x0D,0x99,0x49,0x41,0x1F,0x01,0x09};
void delay(uchar time);
void main()
{
        uchar X,Y;              //定义要显示的变量 X,Y 取值为 0~9 的整数
        SCON = 0X00;            //串行口工作在方式 0
        X = 9;                  //给 X 赋值
        Y = 0;                  //给 Y 赋值
        SBUF = table[X];        //通过串口显示 X
        delay(20);              //延时
        SBUF = table[Y];        //通过串口显示 Y
        while(1);
}
void delay(uchar time)
{
        while(time--);
}
```

例 A.7(汇编语言见例 4.7)

甲机：

```
#include <reg52.h>
#include <intrins.h>

#define NUM 16      //定义发送字节数
bit finish;          //定义发送一个字节成功标志,finishi = 0 表示发送成功
unsigned char data trans[NUM] _at_ 0x40;    //定义发送数组及数组首地址
unsigned char i;

void init(void);

void main(void)
{
        init();
        for(i = 0;i < NUM;i ++)
        {
            finish = 1;         //表示处于发送状态
            ACC = trans[i];     //发送字节送入累加器 A
            TB8 = P;            //校验位送入 TB8
            SBUF = ACC;         //发送字节进入发送缓冲器，开始发送
            while(finish);      //等待发送成功
        }
        ES = 0;
        while(1);
}
void init(void)
{
        SCON = 0xd0;            //置工作方式 3 并允许接收
        TMOD = 0x20;            //置定时器方式 2,自动重装载
        TH1 = 0xfd;             //波特率设置
        TL1 = 0xfd;             //9600@11.0592MHz
        TR1 = 1;                //启动定时器
        EA = 1;                 //CPU 开中断
        ES = 1;                 //允许串行口中断
}
void serial(void) interrupt 4
{
        while(TI)
        {
            TI = 0;             //发送完成,清发送中断
```



```
        }
        while(RI)
        {
             RI = 0;               //清接收中断
             ACC = SBUF;           //接收到的数据送入累加器 A
             if(ACC == 0x00)       //接收正确,清发送标志
             {
                  finish = 0;
             }
             else                  //接收不正确,重新发送
             {
                  ACC = trans[i];
                  TB8 = P;
                  SBUF = ACC;
             }
        }
}
```

乙机:

```
#include <reg52.h>
#include <intrins.h>

#define NUM 16                     //定义接收字节数
bit finish;                        //定义接收一个字节成功标志,finishi = 0 表示接收成功
unsigned char data recei[NUM] _at_ 0x40;     //定义接收数组及数组首地址
unsigned char i;

void init(void);

void main(void)
{
        init();

        for(i = 0;i < NUM;i ++)
        {
             finish = 1;           //表示处于接收状态
             while(finish);        //等待接收成功
        }
        ES = 0;
        while(1);
}
void init(void)
```

```
{
        SCON = 0xd0;            //置工作方式3并允许接收
        TMOD = 0x20;            //置定时器方式2,自动重装载
        TH1 = 0xfd;             //波特率设置
        TL1 = 0xfd;             //9600@11.0592MHz
        TR1 = 1;                //启动定时器
        EA = 1;                 //CPU开中断
        ES = 1;                 //允许串行口中断
}
void serial(void) interrupt 4
{
        while(TI)
        {
            TI = 0;             //发送完成,清发送中断
        }
        while(RI)
        {
            RI = 0;             //清接收中断
            ACC = SBUF;         //接收到的数据送入累加器A
            if(RB8 == P)        //接收正确
            {
                recei[i] = SBUF; //保存接收到的数据
                SBUF = 0x00;     //发送接收成功标志
                finish = 0;      //清接收标志
            }
            else                 //接收到的数据不正确
            {
                SBUF = 0xaa;     //发送接收不正确标志
            }
        }
}
```

附录 B
Proteus 仿真设计

Proteus 软件是英国 Labcenter Electronics 公司出版的 EDA 工具软件，运行于 Windows 操作系统，不但元器件库丰富，而且简单易用。它不仅具有其他 EDA 工具软件的仿真功能，还能仿真 MCU 及外围器件。它是目前很好的仿真 MCU 及外围器件的工具。虽然目前国内推广刚起步，但已受到 MCU 爱好者、从事 MCU 教学的教师、致力于 MCU 开发应用的科技工作者的青睐。Proteus 是世界上著名的 EDA 工具(仿真软件)，从原理图布图、代码调试到 MCU 与外围电路协同仿真，一键切换到 PCB 设计，真正实现了从概念到产品的完整设计。它是目前世界上唯一将电路仿真软件、PCB 设计软件和虚拟模型仿真软件三合一的设计平台，其处理器模型支持 80C51、HC11、PIC10/12/16/18/24/30/DsPIC33、AVR、ARM、8086 和 MSP430 等，2010 年又增加了 Cortex 和 DSP 系列处理器，并持续增加其他系列处理器模型。在编译方面，它也支持 IAR、Keil 和 MPLAB 等多种编译器。

B.1 Proteus 集成开发环境简介

B.1.1 Proteus 简介

Proteus 主要由 ISIS 和 ARES 两部分组成，ISIS 的主要功能是原理图设计及电路原理图的交互仿真，ARES 主要用于印制电路板设计。Proteus 具有以下特点：

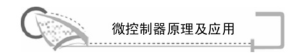
(1) 具有模拟电路仿真、数字电路仿真、MCU 及其外围电路仿真等功能。

(2) Proteus 提供多种激励源，包括直流、正弦、脉冲、分段线性脉冲、音频(使用 wav 文件)、指数信号、音频 FM、数字时钟等，还支持文件形式的信号输入。

(3) Proteus 提供丰富的虚拟仪器，面板操作逼真，如示波器、逻辑分析仪、信号发生器、直流电压/电流表、交流电压/电流表、频率计/计数器、逻辑探头、虚拟终端、SPI 调试器、I^2C 调试器等。

(4) Proteus 提供生动的仿真显示。用色点显示引脚的数字电平，导线以不同颜色表示其对地电压大小，结合动态器件(如电动机、显示器件、按钮)的使用可以使仿真更加直观、生动。

(5) Proteus 提供高级图形仿真功能。基于图标的分析可以精确分析电路的多项指标，包括工作点、瞬态特性、频率特性、传输特性、噪声、失真、傅里叶频谱分析等，还可以进行一致性分析。



(6) Proteus 支持通用外设模型，如字符 LCD 模块、图形 LCD 模块、LED 点阵、LED 七段显示模块、键盘/按键、直流/步进/伺服电动机、RS232 虚拟终端、电子温度计等，其 COMPIM(COM 口物理接口模型)还可以使仿真电路通过 PC 串口和外部电路实现双向异步串行通信。

(7) Proteus 支持多种处理器系统仿真。目前支持的处理器模型有 80C51、HC11、PIC10/12/16/18/24/30/DsPIC33、AVR、ARM、8086、MSP430、Cortex 和 DSP 系列处理器，并持续增加其他系列处理器模型。

(8) Proteus 提供多种调试功能。其具有全速、单步、设置断点等调试功能，同时可以观察各个变量、寄存器的当前状态。

(9) Proteus 支持第三方的软件编译和调试环境，如 Keil C51 µVision4 等软件。

(10) Proteus 具有从原理图设计到 PCB 设计的快速通道。原理图设计完成后，一键便可进入 ARES 的 PCB 设计环境，实现从概念到产品的完整设计。

(11) Proteus 提供先进的自动布局/布线功能。它支持器件的自动/人工布局；支持无网格自动布线或人工布线；支持引脚交换/门交换功能，使 PCB 设计更为合理。

(12) Proteus 具有完整的 PCB 设计功能。它最多可设计 16 个铜箔层，2 个丝印层，4 个机械层(含板边)。提供灵活的布线策略供用户设置，自动设计规则检查，可进行 3D 可视化预览等。

(13) Proteus 可以输出多种格式文件，包括 Gerber 文件的导入或导出，与其他 PCB 设计工具互相转换(如 Protel，现在更名为 Altium Designer)和 PCB 板的设计与加工。

B.1.2 Proteus 安装

了解了 Proteus 的一些基本概况后，下面开始在计算机上搭建 MCU 系统的设计与仿真平台。以单机版安装为例，操作步骤如下。

首先准备 Proteus 安装光盘，双击 Proteus 的安装文件 setup.exe，弹出 Proteus 7 系统安装界面，如图 B-1 所示。



图 B-1 Proteus 7 系统安装界面

单击“Install Proteus”按钮，开始安装 Proteus，弹出如图 B-2 所示的 Proteus 7 安装欢迎界面。





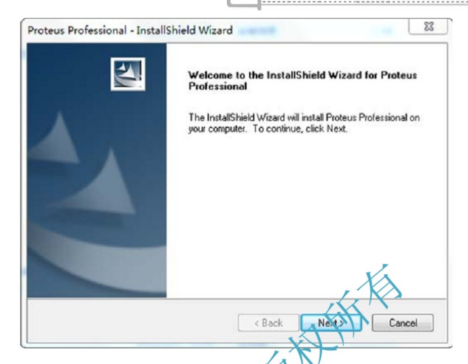


图 B-2　Proteus 欢迎界面

单击“Next”按钮，弹出“License Agreement”对话框，如图 B-3 所示。这里显示了一些用户安装的协议和许可的要求。

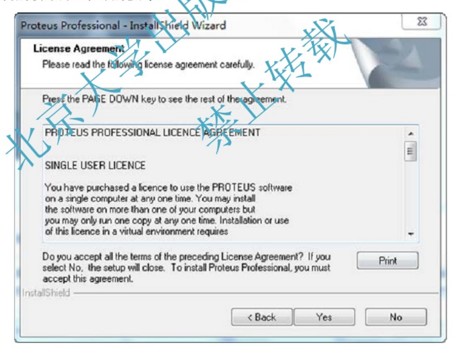


图 B-3　License Agreement 对话框

单击“Yes”按钮，弹出“Setup Type”对话框，如图 B-4 所示。这里显示安装“License Key”的地址，是本机还是服务器，选择默认方式“Use a locally installed License Key”。

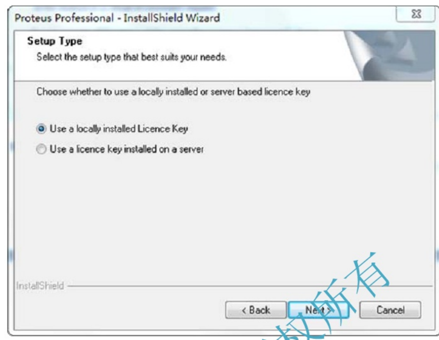


图 B-4 Setup Type 对话框

单击“Next”按钮，如果以前没有安装过 License Key，则弹出“Product License Key”对话框，如图 B-5 所示。

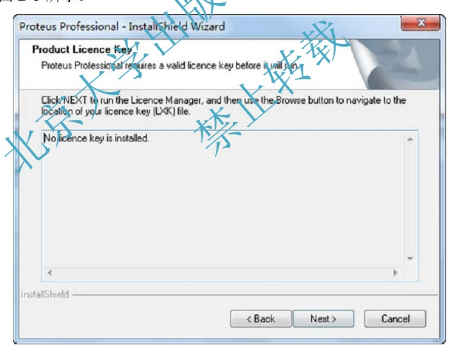


图 B-5 Product License Key 对话框

单击“Next”按钮，弹出“Labcenter License Manager 1.6”窗口，如图 B-6 所示。

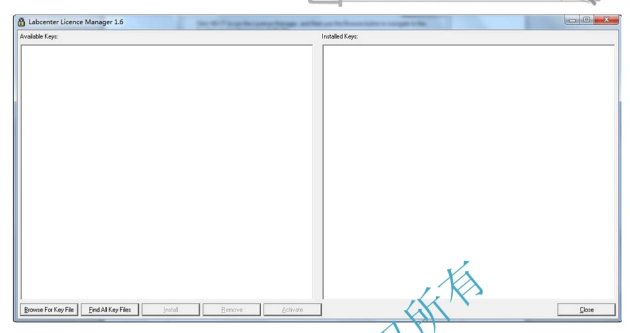


图 B-6　Labcenter License Manager 窗口

单击“Browse For Key File”按钮，寻找 License 文件，选中对应 License 文件，单击“Install”按钮，当 License 显示于右边视窗中，表示 License 安装完毕。单击“Close”按钮，进入显示该 License 的相关信息的窗口，如图 B-7 所示。

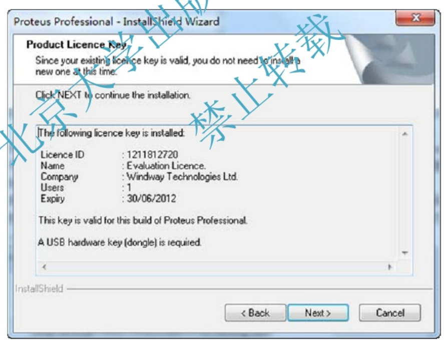


图 B-7　显示 License 的相关信息

单击“Next”按钮，弹出“Choose Destination Location”对话框，如图 B-8 所示。系统默认安装在“C:\Program Files\Labcenter Electronics\Proteus 7 Professional”文件夹下。在这里，单击“Browse”按钮，可以选择安装在其他文件夹下。

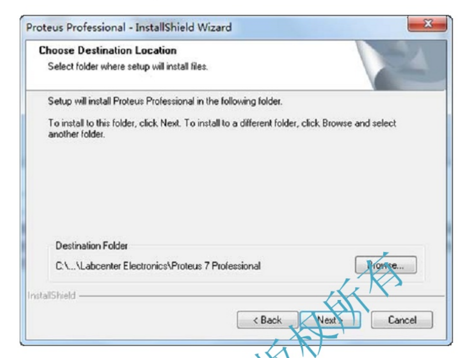


图 B-8　Choose Destination Location 对话框

单击“Next”按钮，弹出“Select Features”对话框，如图 B-9 所示，选择系统默认安装功能即可。

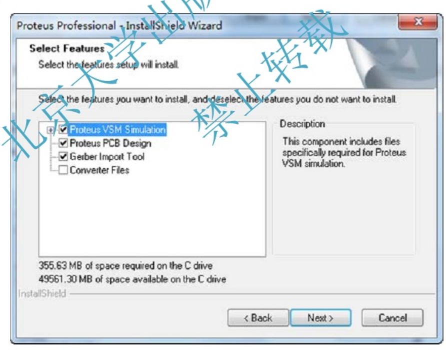


图 B-9　Select Features 对话框

单击“Next”按钮，弹出“Select Program Folder”对话框，如图 B-10 所示。系统默认将所有应用程序安装在开始菜单下的“Proteus 7 Professional”程序文件夹下。在这里，单击编辑框，用户可重新命名程序文件夹的名字。



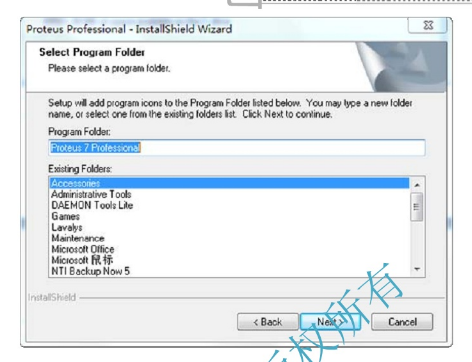


图 B-10　Select Program Folder 对话框

单击“Next”按钮，弹出“Setup Status”对话框，如图 B-11 所示。等待程序安装完毕或者单击“Cancel”按钮取消程序安装。

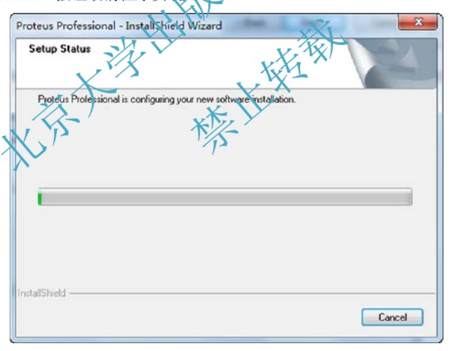


图 B-11　Setup Status 对话框

安装过程中，出现 USB 硬件加密狗驱动安装的提示，此时确保加密狗未插在机器中，如图 B-12 所示。



图 B-12　提示不要将加密狗插在 USB 插槽中

单击“确定”按钮，显示加密狗驱动安装完成，提示现在可以将 Proteus USB 加密狗插入到空闲的 USB 插槽中，如图 B-13 所示。插入加密狗，红色指示灯亮，安装完成。

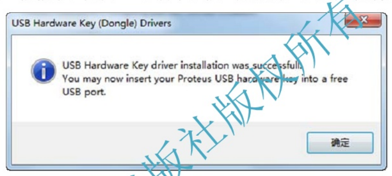


图 B-13　提示将加密狗插在空闲的 USB 插槽中

程序继续安装，直到弹出“InstallShield Wizard Complete”对话框，如图 B-14 所示，提示程序安装完毕。默认单击“Finish”按钮，显示 Proteus 7 的新特性；如果不想浏览新特性，可单击复选框，取消显示。

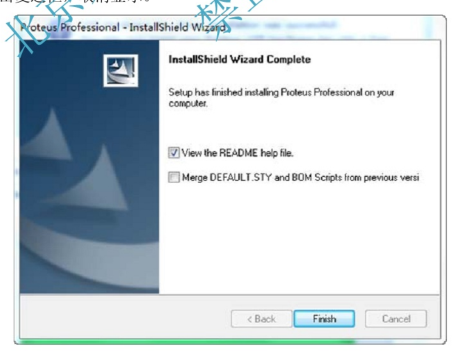


图 B-14　InstallShield Wizard Complete 对话框



最后，单击“Finish”按钮，便可以结束 Proteus 集成开发环境的安装。

B.1.3　Proteus 7 集成开发环境界面

安装完成后，会在“开始”程序里增加“Proteus 7 Professional”程序文件夹。单击此文件夹下的“ISIS 7 Professional”程序项，即可启动“PROTEUS DESIGN SUITE ISIS SCHEMATIC CAPTURE”，启动画面如图 B-15 所示。启动完成后，弹出如图 B-16 所示窗口。



图 B-15　ISIS 7 启动界面

图 B-16　ISIS 调用设计文件例子选项

单击“Yes”按钮，弹出调用设计文件例子的窗口，单击“No”按钮，可以建立自己的设计文件。可选中复选框，以便下次不再显示此对话框。无论选择“Yes”或“No”，都会显示打开 GL 图形驱动的提示，如图 B-17 所示。选中复选框，可不再显示此提示信息。

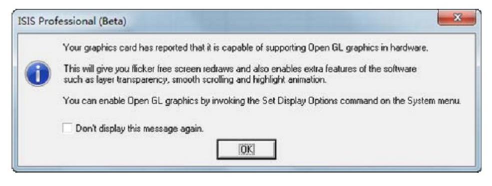


图 B-17　GL 图形驱动提示

单击“OK”按钮，显示如图 B-18 所示的 Proteus 仿真界面。

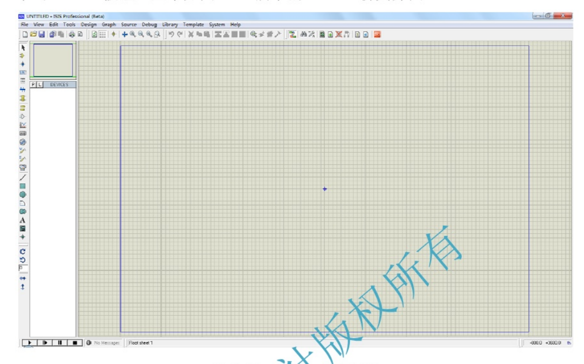


图 B-18　Proteus 仿真界面

B.1.4　Proteus ISIS 菜单命令

Proteus ISIS 的菜单栏提供了项目操作、编辑操作、编译调试及帮助等各种常用操作。所有的操作基本上都可以通过菜单命令来实现。为了快速执行 Proteus ISIS 的许多功能，有些菜单命令在工具栏上还有工具条。为了更快速执行一些功能，Proteus ISIS 提供了比工具栏上的工具条更为快捷的操作，即快捷键。下面就菜单命令、工具条、快捷键分别进行介绍。

1. File 菜单

File 菜单和标准 Windows 软件的 File 菜单类似，包括常用的文件功能。File 菜单各个命令的功能见表 B-1。

表 B-1　File 菜单

菜单命令	工具条	快捷键	功能说明
New Design…			创建一个新的设计
Open Design…		Ctrl+O	打开一个已存在的设计
Save Design		Ctrl+S	保存当前打开的设计
Save Design as…			当前设计另存为
Save Design as Template…			将当前设计保存为模板
Windows Explorer…			打开资源浏览器



续表

菜单命令	工具条	快捷键	功能说明
Import Bitmap…			在当前设计中插入位图
Import Section…			将局部文件导入 ISIS 中
Export Section…			将当前选中的对象导出为局部文件
Export Graphics			将选中对象导出为指定格式的图形文件
Mail To…			邮寄给…
Print…			打印当前文件
Printer Setup…			设置打印机
Printer Information			打印机信息
Set Area			打印选中的区域
1 .. 4			列出最近打开的设计文件
Exit			退出 ISIS

2. View 菜单

View 菜单提供编辑区的显示、定位、缩放及栅格调整等操作功能。View 菜单各个命令的功能见表 B-2。

表 B-2 View 菜单

菜单命令	工具条	快捷键	功能说明
Redraw		R	刷新显示内容
Grid		G	设定是否显示网格点
Origin		O	重新定义原点
X cursor		X	鼠标样式切换
Snap 10th		Ctrl+F1	设定捕捉单位为 10 毫英寸
Snap 50th		F2	设定捕捉单位为 50 毫英寸
Snap 100th		F3	设定捕捉单位为 100 毫英寸
Snap 0.5in		F4	设定捕捉单位为 500 毫英寸
Pan		F5	以鼠标所在点为中心显示编辑区
Zoom In		F6	放大编辑区
Zoom Out		F7	缩小编辑区
Zoom All		F8	显示整个编辑区
Zoom to Area			显示局部编辑区
Toolbars…			设定是否显示工具条

3. Edit 菜单

Edit 菜单提供了编辑区常用的编辑操作命令。Edit 菜单各个命令的功能见表 B-3。

表 B-3 Edit 菜单

菜单命令	工具条	快捷键	功能说明
Undo		Ctrl+Z	撤销最后的操作
Redo		Ctrl+Y	恢复最后的操作
Find and Edit Component…		E	查找并编辑元件属性
Cut to clipboard			剪切到剪贴板
Copy to clipboard			复制到剪贴板
Paste from clipboard			从剪贴板粘贴
Align		Ctrl+A	对齐操作
Send to back		Ctrl+B	置于底层
Bring to front		Ctrl+F	置于顶层
Tidy			清除元件列表中未用的元件

4. Tools 菜单

Tools 菜单提供了编辑区常用的工具操作命令。Tools 菜单各个命令的功能见表 B-4。

表 B-4 Tools 菜单

菜单命令	工具条	快捷键	功能说明
Real Time Annotation	U1	Ctrl+N	实时标注
Wire Auto Router		W	自动布线
Search and Tag…		T	查找并选中
Property Assignment Tool…		A	属性分配工具
Global Annotator…			全局注释
ASCII Data Import…			导入 ASCII 数据
Bill of Materials			元件清单
Electrical Rule Check…			电气规则检查
Netlist Compiler…			编译网络标号
Model Compiler…			编译模型
Set filename for PCB Layout…			为 PCB 图层设置文件名
Netlist to ARES	ARES	Alt+A	将网络标号导入 PCB
Backannotate from ARES			从 PCB 设计返回原理图设计



5. Design 菜单

Design 菜单提供了原理图设计常用的操作命令。Design 菜单各个命令的功能见表 B-5。

表 B-5　Design 菜单

菜单命令	工具条	快捷键	功能说明
Edit Design Properties…			编辑设计属性
Edit Sheet Properties…			编辑图纸属性
Edit Design notes…			编辑设计注释
Configure Power Rails…			配置电源
New Sheet			新建原理图
Remove Sheet			删除原理图
Previous Sheet		Page-Up	转到上一张原理图
Next Sheet		Page-Down	转到下一张原理图
Goto Sheet			转到原理图
Design Explorer		Alt+X	设计浏览器

6. Graph 菜单

Graph 菜单提供了设计常用的图形编辑、分析操作命令。Graph 菜单各个命令的功能，见表 B-6。

表 B-6　Graph 菜单

菜单命令	工具条	快捷键	功能说明
Edit Graph			编辑图形
Add Trace…		Ctrl+T	增加跟踪曲线
Simulate Graph		Space	仿真图形
View Log		Ctrl+V	查看日志
Export Data			导出数据
Clear data			清除数据
Conformance Analysis(All Graphs)			(全图)一致性分析
Batch Mode Conformance Analysis…			批处理一致性分析

7. Source 菜单

Source 菜单提供了源代码编辑操作命令。Source 菜单各个命令的功能见表 B-7。

表 B-7　Source 菜单

菜单命令	工具条	功能说明
Add/Remove Source files…		添加/删除源程序
Define Code Generation Tools…	[icon]	定义代码生成工具
Setup External Text Editor…		设置外部文本编辑器
Build All		编译全部代码

8. Debug 菜单

Debug 菜单中的命令大多用于仿真调试过程中，提供了断点、调试方式、资源查看及配置等功能。Debug 菜单各个命令的功能见表 B-8。

表 B-8　Debug 菜单

菜单命令	工具条	快捷键	功能说明
Start/Restart Debugging	[icon]	Ctrl+F12	启动/重新启动调试
Pause Animation	[icon]	Pause	暂停仿真
Stop Animation	[icon]	Shift+Pause	停止仿真
Execute	[icon]	F12	全速运行
Execute Without Breakpoints		Alt+12	忽略断点连续运行
Execute for Specified Time			按指定的时间段运行
Step Over	[icon]	F10	单步执行程序，跳过子程序
Step Into	[icon]	F11	单步执行程序，遇到子程序则进入
Step Out	[icon]	Ctrl+F11	程序执行到当前函数结束
Step To	[icon]	Ctrl+F10	程序执行到光标所在行
Animate		Alt+F11	以动画形式执行程序
Reset Popup Windows			复位弹出窗口
Reset Persistent Model Data			复位永久模型数据
Configure Diagnostics…	[icon]		配置诊断窗口
Use Remote Debug Monitor			使用远程调试器
Tile Horizontally	[icon]		水平排列窗口
Tile Vertically	[icon]		垂直排列窗口
Simulation Log			仿真日志
Watch Window			观察窗口



续表

菜单命令	工具条	快捷键	功能说明
8051 CPU Registers			8051 CPU 寄存器窗口
8051 CPU SFR Memory			8051 CPU SFR 寄存器窗口
8051 CPU Internal(IDATA) Memory			8051 CPU 内部数据寄存器窗口
8051 CPU EEPROM			8051 CPU 程序存储器窗口
8051 CPU Source Code			8051 CPU 源代码窗口
8051 CPU Variables			8051 CPU 变量窗口

9. Library 菜单

Library 菜单提供了设计常用的元件库操作命令。Library 菜单各个命令的功能见表 B-9。

表 B-9　Library 菜单

菜单命令	工具条	快捷键	功能说明
Pick Device/Symbol…		P	添加元件和符号
Make Device…			制作元件
Make Symbol…			创建图标
Packaging Tool…			封装工具
Decompose			释放元件
Compile to Library…			编译元件库
Autoplace Library…			自动放置元件库
Verify Packaging…			验证封装
Library Manager…			元件库管理器

10. Template 菜单

Template 菜单提供了设计常用的模板操作命令。Template 菜单各个命令的功能见表 B-10。

表 B-10　Template 菜单

菜单命令	工具条	功能说明
Goto Master Sheet		显示主原理图
Set Design Defaults…		编辑默认设计选项
Set Graph Colours…		编辑图形颜色
Set Graphics Styles…		编辑图形样式



续表

菜单命令	工具条	功能说明
Set Text Styles…		编辑全局文本样式
Set Graphics Text…		编辑图形文本格式
Set Junction Dots…		编辑节点样式
Load Styles from Design…		从已有设计中调用模板样式
Apply Default Template…		使用默认模板

11. System 菜单

System 菜单提供了设计常用的系统设置命令。System 菜单各个命令的功能见表 B-11。

表 B-11　System 菜单

菜单命令	工具条	功能说明
System Info…		显示软件信息
Check for Updates…		检查软件更新
Text Viewer		文本观察器
Set BOM Scripts…		设置元件清单格式
Set Display Options…		设置显示选项
Set Environment…		设定系统环境选项
Set Keyboard Mapping…		设置键盘快捷方式
Set Paths…		设置路径
Set Property Definitions…		设置默认属性定义
Set Sheet Sizes…		编辑图纸大小
Set Text Editor…		设置文本编辑器选项
Set Animation Options…		设置仿真电路选项
Set Simulator Options…		设置仿真器选项
Restore Default Settings		恢复默认设置

另外，Help 菜单提供了一些帮助信息，与其他软件类似，这里不再具体介绍。

B.1.5　Proteus ISIS 工具栏

Proteus 工具栏分为：命令工具栏、模式选择工具栏、方向工具栏、仿真工具栏。



1. 命令工具栏

：复制选中的块对象
：移动选中的块对象
：旋转选中的块对象
：删除选中的块对象
：返回父设计页
：元件清单

2. 模式选择工具栏

：选择对象方式
：选择元器件
：放置连接点
：放置连线标签
：放置文本
：画总线
：画子电路
：元件引脚
：仿真图表
：磁带记录器模式
：信号发生器
：电压探针
：电流探针
：虚拟仪表
：画各种直线
：画各种方框
：画各种圆
：画各种圆弧
：画各种多边形
：写各种文字
：画符号
：画原点

3. 方向工具栏

：顺时针旋转
：逆时针旋转
：水平翻转
：垂直翻转

4. 仿真工具栏



：开始仿真

：单步仿真

：暂停仿真

：停止仿真

B.2　Proteus 仿真实例

Proteus 强大的 MCU 系统设计及仿真功能，使它成为 MCU 系统应用开发和改进的重要手段之一。MCU 系统设计及仿真的全过程都是在计算机上通过 Proteus 来完成的。要创建一个应用设计，需要按下列步骤操作。

(1) Proteus 电路设计：在 ISIS 平台上进行 MCU 系统的电路设计、选择元器件、接插件、连接电路和电气检测等。

(2) Proteus 源程序设计和生成目标代码文件：在 ISIS 平台上进行 MCU 系统程序设计、编辑、汇编代码编译、代码调试，最后生成目标代码文件(*.HEX)。

(3) Proteus 仿真：在 ISIS 平台上将目标代码文件加载到 MCU 系统中，并实现 MCU 系统的实时交互、协同仿真。它基本上反映了 MCU 系统的实际运行状况。

下面通过一个实例，详细介绍如何在 Proteus 集成开发环境中设计并仿真 MCU 系统。

【例 B-1】以 AT89C51 为例，设计一个用按键控制的跑马灯，假设晶振频率为 12MHz。当按下按键的时候，8 个 LED 依次点亮；当再次按下按键时，8 个 LED 停止依次点亮。

B.2.1　Proteus 电路设计

单击开始菜单中的“Proteus 7 Professional”程序文件夹，再单击“ISIS 7 Professional”程序项，启动 Proteus 集成开发环境。

1. 鼠标操作

Proteus 的鼠标操作与一般的 EDA 软件不同，需要一个适应的过程。下面介绍各种对象的鼠标操作方式。

放置对象：单击鼠标左键，放置元器件、连线等。

选中对象：单击鼠标右键，选择元器件、连线和其他对象，此时选中的操作对象以高亮红色(系统默认颜色)显示。

删除对象：双击鼠标右键，删除元器件、连线等。

选择块对象：按住鼠标左键(或右键)拖出方框，选中方框中的多个元器件及其连线。

编辑对象：先单击鼠标右键，后单击鼠标左键(或者单击弹出式菜单中的 Edit Properties)，编辑元器件属性。

移动对象：先单击鼠标右键选中对象，按住鼠标左键移动，拖动元器件、连线。

缩放对象：滚动鼠标中键，以鼠标停留点为中心，缩放电路。



2. 新建设计文件

单击菜单中的“File - New Design”，出现选择模板窗口，如图 B-19 所示。其中横向图纸为 Landscape，纵向图纸为 Portrait，DEFAULT 为默认模板。选中模板 DEFAULT，再单击“OK”按钮，则选定了模板 DEFAULT。

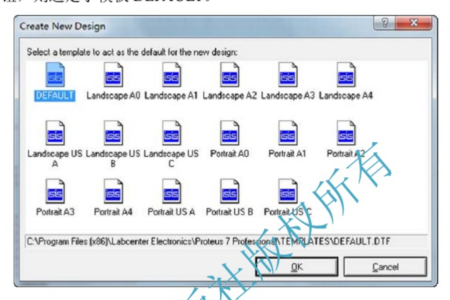


图 B-19　图纸模板选择

单击按钮，弹出如图 B-20 所示的“Save ISIS Design File”窗口。在文件名框中输入 START(实例的文件名)后，再单击“保存”按钮，则完成新建设计文件操作，其后缀自动为.DSN，即 START.DSN。

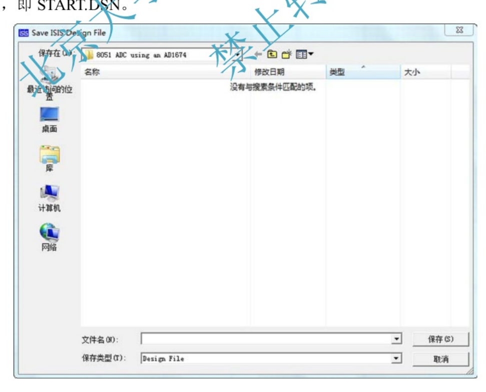


图 B-20　保存 ISIS 设计文件

当启动 Proteus 进入 ISIS 系统后，自动出现一个空白设计，模板默认为 DEFAULT，它的文件名在窗口顶端的标题栏，为未命名 Untitled。可单击按钮，对新建设计文件命名。

3. 设定绘图纸大小

当前的用户图纸大小为默认 A4：长×宽为 10in×7in。若要改变图纸大小，单击菜单中的“System - Set Sheet Size”，弹出如图 B-21 所示的窗口，在窗口可以选择 A0～A4 其中之一，也可以自己设置图纸大小，选中 User 右边的复选框，再按需要更改右边的长和宽的数据。本例图纸大小采用默认 A4。

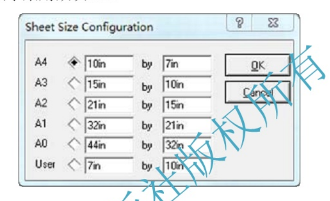


图 B-21　图纸大小设置窗口

4. 选取元器件并添加到对象选择器中

本实例采用的元器件为：MCU(AT89C51)、晶振(CRYSTAL)、瓷片电容(CAP)、电解电容(CAP-ELEC)、电阻(RES)、按钮(BUTTON)、发光二极管(LED-RED)。

单击图 B-22 中的“P”按钮，弹出如图 B-23 所示的选取元器件窗口。



图 B-22　单击“P”按钮

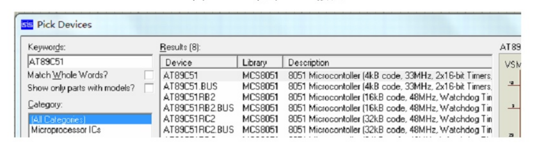


图 B-23　元器件列表

在其左上角 Keywords(关键字)一栏中输入元器件名称 AT89C51，则出现与关键字匹配的元器件列表。选中并双击 AT89C51 所在行或单击 AT89C51 所在行后，再单击“OK”按



钮，便将器件 AT89C51 加入到 ISIS 对象选择器中。按此操作方法完成其他元器件的选取。关键字相应为 CAP、CAP-ELEC 等。被选取的元器件都加入到 ISIS 对象选择器中，如图 B-24 所示。

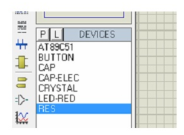


图 B-24　选取的元器件均加入到 ISIS 对象选择器中

上述的选取方法称“关键字查找法”。关键字可以是对象的名称(全名或其部分)、描述、分类、子类，甚至是对象的属性值。若与搜索结果相匹配的元器件太多，可以通过限定分类、子类来缩小搜索范围，再做取舍，如要找 5.1kΩ电阻，以 5.1k 为关键字查找，再在列表中进一步选择，如图 B-25 所示。

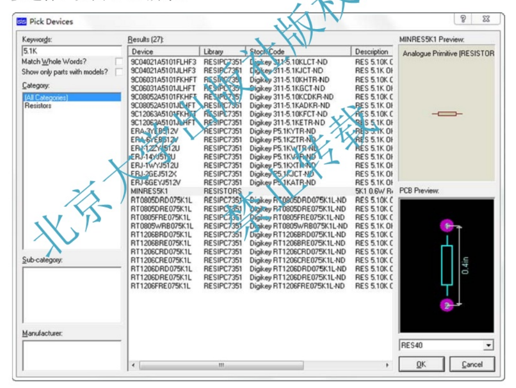


图 B-25　选取元器件窗口

还有一种“分类法”，以元器件所属大类、子类甚至生产厂家为条件一级一级地缩小范围进行查找。在具体操作时，常将这两种方法结合使用。

5. 网格单位

图 B-26 所示默认的网格单位是 100th，这也是移动元器件的步长单位，可根据需要改变这一单位。单击菜单“View”(查看)再单击所要的网格单位即可。如图 B-26 所示，选项左侧复选框打“√”的项为选中项，也可按快捷键 F2、F3、F4 或 Ctrl+F1 设置相应的网格单位。

6. 放置、移动、旋转元器件

单击 ISIS 对象选择器中的元器件名，蓝色条出现在该元器件名上，并且该元器件显示在对象预览窗口，单击转向按钮、、、、，可在对象预览窗口观察到元器件方向的变化。把鼠标指针移到编辑区某位置后，单击就可放置元器件于该位置，每单击一次，就放一个元器件。要移动元器件，先单击鼠标右键使元器件处于选中状态(即高亮状态)，再按住鼠标左键拖动，元器件就跟随指针移动，如图 B-27 所示，到达目的地后，松开鼠标左键即可。要调整元器件方向，先将指针指在元器件上单击鼠标右键选中，再单击弹出式菜单相应的转向按钮、、、、。若多个对象一起移动或转向，选择相应的块操作命令。

	Snap 10th	Ctrl+F1
	Snap 50th	F2
✔	Snap 100th	F3
	Snap 0.5in	F4

图 B-26　网格单位选择

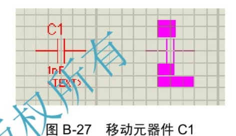


图 B-27　移动元器件 C1

通过放置、移动、旋转元器件操作，可将各元器件放置在 ISIS 编辑区中的合适位置，如图 B-28 所示。

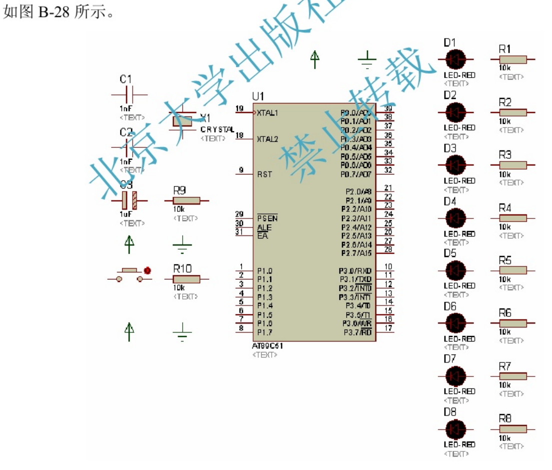


图 B-28　本例用到的元器件放置到 ISIS 编辑区的合适位置



7. 放置电源、地(终端)

放置 POWER(电源)操作：单击模式选择工具栏中的终端按钮，在 ISIS 对象选择器中单击 POWER，如图 B-29 所示，再在编辑区要放置电源的位置单击完成放置，如图 B-28 所示。放置 GROUND(地)的操作与其类似。

图 B-29　放置终端符号

8. 电路图布线

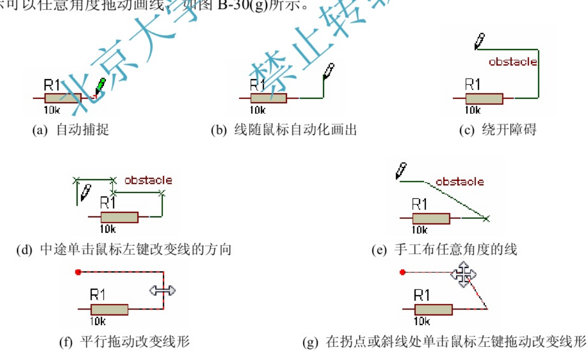
系统默认自动捕捉和自动布线(命令工具栏)有效。相继单击元器件引脚间、线间等要连线的两处，会自动生成连线。

(1) 自动捕捉：在自动捕捉有效的情况下，当光标靠近引脚末端或线时该处会自动感应出现一个“虚框”和“笔”，表示从此点可以单击画线，如图 B-30(a)所示。

(2) 自动布线：在前一指针着落点和当前点之间会自动预画线，它可以是带直角的线，如图 B-30(b)所示。在引脚末端选定第一个画线点后，随指针移动自动有预画细线出现，当遇到障碍时，会自动绕开障碍，如图 B-30(c)所示。这正是智能绘图的表现。

(3) 手工调整线形：要进行手工直角画线，直接在移动鼠标的过程中单击鼠标左键即可，如图 B-30(d)所示。若要手工任意角度画线，在移动鼠标的过程中按住 Ctrl 键，移动指针，预画线自动随指针呈任意角度，确定后单击鼠标左键即可，如图 B-30(e)所示。

(4) 移动画线、改变线形：选中要改变的画线，指针靠近画线，出现捕捉标志“虚框”，按下鼠标左键，若出现双箭头，表示可沿垂直于该线的方向移动。此时拖动鼠标，就近的线会跟随移动，图 B-30(f)所示的为水平拖动；按住拐点或斜线上任意一点，会出现四箭头，表示可以任意角度拖动画线，如图 B-30(g)所示。



(a) 自动捕捉　(b) 线随鼠标自动化画出　(c) 绕开障碍

(d) 中途单击鼠标左键改变线的方向　(e) 手工布任意角度的线

(f) 平行拖动改变线形　(g) 在拐点或斜线处单击鼠标左键拖动改变线形

图 B-30　画线及其移动、改变形状

9. 设置、修改元器件的属性

Proteus 库中的元器件都有相应的属性，要设置、修改它的属性，可在 ISIS 编辑区中的元器件上单击鼠标右键，再单击鼠标左键(或者在其弹出式菜单中单击 Edit Properties)打开其属性窗口，这时可在属性窗口中设置、修改它的属性。例如，发光管的限流电阻 R1，先在 R1 上单击鼠标右键，再单击鼠标左键打开其属性窗口如图 B-31 所示，图中已将电阻值 10kΩ修改为 300Ω。其他元器件属性值修改结果如图 B-32 所示。

图 B-31　设置限流电阻位 300Ω

图 B-32　编辑完成的简单电路图



10. 电气规则检测

设计电路完成后，单击命令工具栏中的电气规则检查按钮，会出现检查结果窗口，如图 B-33 所示。窗口前面是一些文本信息，接着是电气检查结果列表，若有错，会有详细说明。当然，也可通过菜单操作“Tools - Electrical Rule Check…”完成电气规则检测。



图 B-33　电气检测窗口

B.2.2　源程序设计

1. 添加源程序文件

单击 ISIS 菜单“Source”(源程序)，弹出下拉菜单如图 B-34 所示。

图 B-34　添加源程序菜单

单击“Add/Remove Source files…”(添加/删除源程序)选项，弹出如图 B-35 所示窗口，单击 Code Generation Tool(目标代码生成工具)下方框中按钮，弹出下拉菜单，选择代码生成工具 ASEM51(51 系列及其兼容系列汇编器)。

若 Source Code Filename(源程序文件名)下方框中没有期望的源程序文件，则单击“New”(新建)按钮，弹出如图 B-36 所示的对话框，在对话框文件名框中输入新建源程序文件名 start.asm(实例源程序名)后，单击“打开”按钮，会弹出图 B-36 中的提示框，单击“是”按钮，新建的源程序文件就添加到图 B-35 中的 Source Code Filename 下方框中，如图 B-35 所示。同时在菜单 Source 中也出现源程序文件 start.asm，如图 B-37 所示。

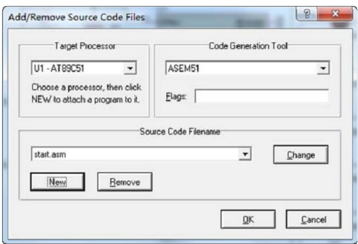


图 B-35　Add/Remove Source Code Files…窗口

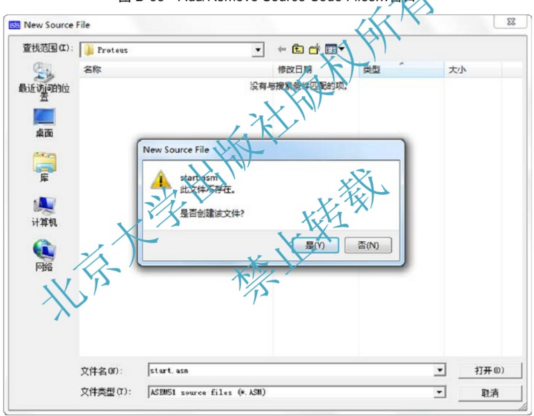


图 B-36　新建源程序文件

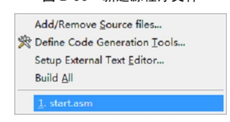


图 B-37　源程序文件加载到 ISIS

2. 编写编辑源程序

单击菜单“Source - start.asm”，如图 B-37 所示，在图 B-38 的源程序编辑窗口中编辑源程序。编辑无误后，单击按钮存盘，文件名就是 start.asm。



图 B-38　源程序编辑窗口及编辑完成的源程序

B.2.3　生成目标代码文件

1. 目标代码生成工具设置

如果首次使用某一编译器，则需设置代码产生工具，单击菜单“Source - Define Code Generation Tools”，如图 B-39 所示。

其中，Code Generation Tool(代码生成工具)设置为 ASEM51；Make Rules(生成规则)中，Source Extn(源程序扩展名)设置为 ASM；Obj Extn(目标代码扩展名)设置为 HEX；Command Line(命令行)设置为%1；Debug Data Extraction(调试数据提取)中，List File Extn 设置为 LST。

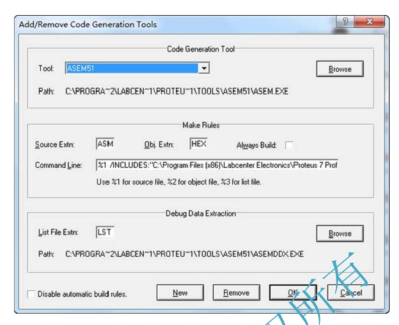


图 B-39　目标代码生成工具设置

2. 汇编编译源程序、生成目标代码文件

单击菜单“Source – Build All”(全编译、汇编)，编译结果在弹出的编译日志窗口中如图 B-40 所示，无错则生成目标代码文件。对 ASEM51 系列及其兼容 MCU 而言，目标代码文件格式为*.HEX。这里生成的是实例目标代码文件 START.HEX。若有错，则可根据编译日志提示来调试源程序，直至无错生成目标代码文件为止。

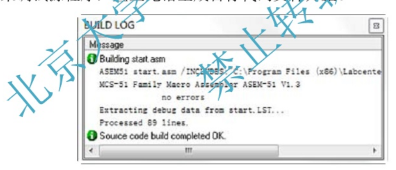


图 B-40　编译日志窗口

B.2.4　加载目标代码文件、设置时钟频率

单击鼠标右键选中 ISIS 编辑区中的 AT89C51，再单击鼠标左键打开其属性窗口，在其中的 Program File 右侧框中输入目标代码文件(目标代码与 DSN 文件在同一文件夹下，直接输入代码文件名即可，否则要写出完整的路径。或单击本栏打开按钮，选取目标文件)，这里是本实例的 start.HEX，如图 B-41 所示。再在 Clock Frequency(时钟频率)栏中设置 12MHz，仿真系统则以 12MHz 的时钟频率运行。因为运行时钟频率以 MCU 属性设置中的 Clock Frequency 为准，所以在编辑区设计以仿真为目标的 80C51 系列 MCU 系统电路时，可以略去 MCU 振荡电路。另外，对 80C51 系列 MCU 而言，复位电路也可略去，EA 控制引脚也可悬空。但如果要进行电气规则检测，则不可略去。

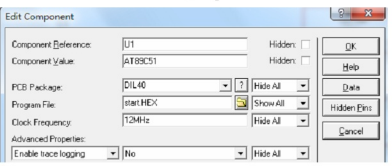


图 B-41　加载目标代码文件

B.2.5　MCU 系统的 Proteus 交互仿真

直接单击仿真按钮中的运行按钮，则会全速仿真，此时全部 8 个 LED 都熄灭。可用鼠标单击图 B-42 中的按钮，实现交互仿真。单击第一次按钮，则 8 个 LED 依次点亮；再次单击按钮，LED 停止循环点亮，单击按钮前点亮的 LED 保持常亮。如此循环，LED 循环点亮，或停止循环。若单击停止仿真按钮，则终止仿真。若进一步调试，可通过“DEBUG”菜单进行。

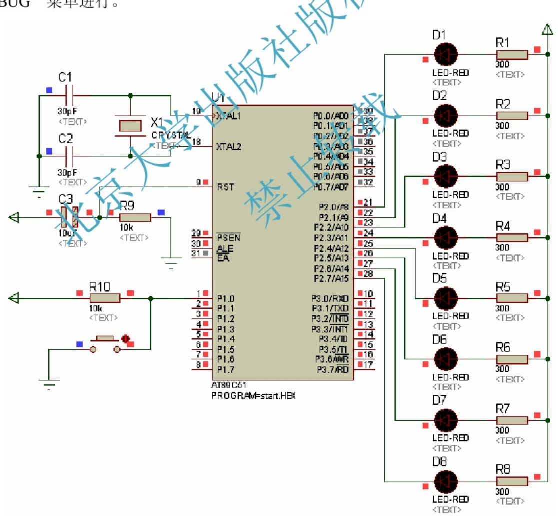


图 B-42　“简单实例”全速仿真片段

B.2.6 MCU 系统的 Proteus 源代码调试仿真

1. 调试菜单及调试窗口

单击按钮 ▶ ，启动仿真。在全速运行时不显示调试窗口，单击暂停按钮 ⏸ ，弹出源程序调试窗口，如图 B-43 所示。

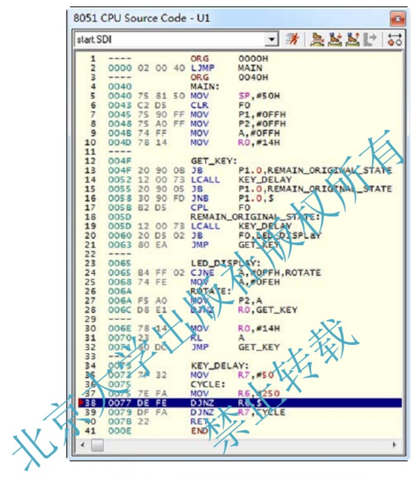


图 B-43 暂停仿真时弹出源代码调试窗口

若未出现，再单击菜单“Debug”，在弹出下拉菜单中，如图 B-44 所示，单击选择“8051 CPU Source Code - U1”，即可显示源代码调试窗口如图 B-43 所示，光条停在下一条要执行的指令行 DJNZ R6,$。在调试窗口右上角有 5 个调试按钮、、、、，其功能见菜单“Debug”的说明。要查看其他窗口，在图 B-44 的相应调试项所在行上单击，该项前出现“ √ ”，则表示已打开相应的窗口。

在调试窗口中单击鼠标右键可弹出菜单，如图 B-45 所示。其中，有快速移动光条的 Goto 命令；有断点操作的命令；有在指令行显示行号、地址等信息的命令；还有设置显示字体、颜色等的命令。在操作时可选择菜单相应命令行单击或是操作相应的快捷键，如设置、清除断点按 F9 键来快速操作。图 B-45 中“Display Line Numbers”（显示行号）、“Display Addresses”（显示地址）、“Display Opcodes”（显示操作码命令行）及“Fix-up Breakpoints On Load”（加载时固定断点）前出现“ √ ”，表示相应显示内容已打开。

Start/Restart Debugging	Ctrl+F12
Pause Animation	Pause
Stop Animation	Shift+Pause
Execute	F12
Execute Without Breakpoints	Alt+F12
Execute for Specified Time	
Step Over	F10
Step Into	F11
Step Out	Ctrl+F11
Step To	Ctrl+F10
Animate	Alt+F11
Reset Popup Windows	
Reset Persistent Model Data	
Configure Diagnostics...	
Use Remote Debug Monitor	
Tile Horizontally	
Tile Vertically	
1. Simulation Log	
2. Watch Window	
3. 8051 CPU Registers - U1	
4. 8051 CPU SFR Memory - U1	
5. 8051 CPU Internal (IDATA) Memory - U1	
6. 8051 CPU Source Code - U1	

图 B-44　调试菜单

Dissassembly	Ctrl+D
Goto Line...	Ctrl+L
Goto Address...	
Find...	Ctrl+F
Find Again	Ctrl+G
Toggle (Set/Clear) Breakpoint	F9
Enable All Breakpoints	
Disable All Breakpoints	
Clear All Breakpoints	Ctrl+F9
✓ Fix-up Breakpoints On Load	
✓ Display Line Numbers	
✓ Display Addresses	
✓ Display Opcodes	
Set Font...	
Set Colours...	

图 B-45　源代码调试窗口的弹出式菜单

2. 存储器窗口

从调试菜单中可看出 CPU 源代码就在调试窗口中，另外还有 3 个存储器窗口。

1) MCU 寄存器窗口

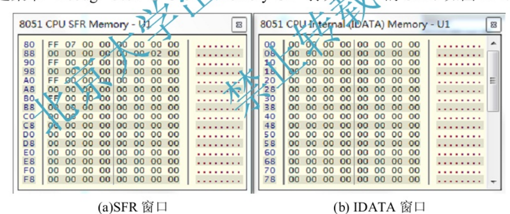
通过菜单“Debug - 8051 CPU Registers-U1”打开 MCU 寄存器窗口，如图 B-46 所示。其中除有 R0～R7 外，还有常用的 SFR，如 SP、PC、将要执行的指令等。在本窗口内右击，弹出可设置本窗口的快捷菜单，如图 B-46 所示。



图 B-46 寄存器窗口

2) MCU 的 SFR 窗口

通过菜单“Debug - 8051 CPU SFR Memory-U1”打开 MCU 的 SFR，如图 B-47(a)所示。



(a)SFR 窗口　　(b) IDATA 窗口

图 B-47 SFR 和 IDATA 窗口

3) MCU 的 IDATA 窗口

通过菜单“Debug - 8051 CPU Internal (IDATA) Memory-U1”打开 MCU 的 IDATA 窗口，如图 B-47(b)所示。

若要查看寄存器 P0、P1 的内容，既可从 MCU 寄存器窗口中查看(图 B-46)，也可从 SFR 寄存器窗口中查看(图 B-47(a))。

在 SFR、IDATA 窗口中单击鼠标右键可弹出设置本窗口的菜单，如图 B-48 所示。由此可用 Goto 命令方便地快速移动显示内容。还可设置存储单元内容的显示类型、显示格式，设置显示字体、颜色等。



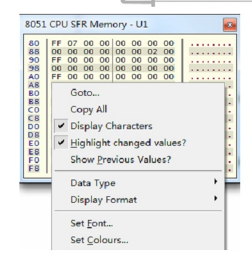


图 B-48　存储器窗口的弹出式菜单

3. 鼠标操作断点

单击按钮，启动仿真。在全速执行时不显示代码窗口及寄存器窗口，单击按钮，可使各调试窗口显示出来。也可在适当的位置设置断点，使运行暂停，观察各窗口。有效断点以实心圆“●”标示，无效断点以空心圆“○”标示。

1) 设置断点

单击要设置断点的行后，出现光条，再单击调试窗口右上角中的按钮；或者在要设置断点的行双击鼠标左键。

2) 取消断点

在有效断点的行上双击鼠标左键，或单击有效断点行，再单击按钮。

3) 清除断点

在无效的断点行上双击鼠标左键，或单击无效行，再单击按钮。若连续双击鼠标左键或是连续单击按钮，则可在设置断点、取消断点、清除断点之间切换。如图 B-49 所示，当前断点在第 14 行：LCALL KEY_DELAY，代码首地址为 0052H，机器码为 120073H。MCU 源代码调试窗口中的第 1 列为行号，第 2 列为命令行首地址，第 3 列为机器码。

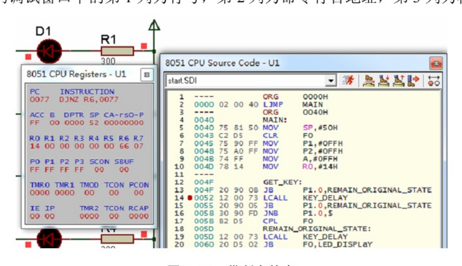


图 B-49　带断点仿真

4. 观察窗口应用

虽然通过调试菜单可以打开 MCU 的各个存储器窗口，来查看各存储单元的内容，但窗口较分散；它们同时出现在计算机的屏幕上，也太拥挤。而且这些窗口在连续仿真运行时不会显示，只在暂停时才显示，这不利于即时观察。“Watch Window” (观察窗口)可克服上述缺点，它与仿真电路一同实时显示，且其中的观察对象可以是 MCU 内 RAM 中任一单元。通过菜单“Debug - Watch Window”打开空白的观察窗口，如图 B-50 所示。

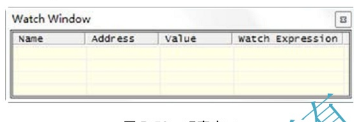


图 B-50　观察窗口

在观察窗口内单击鼠标右键，弹出快捷菜单如图 B-51 所示。由该菜单可添加、删除观察项，可设置观察项的数据类型，设置显示观察项的地址、变化前的值，设置观察窗口的字体、颜色等。

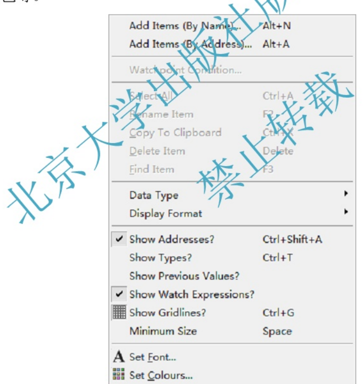


图 B-51　观察窗口的弹出式菜单

1) 添加观察项

(1) 以观察项名称添加观察项。单击观察窗口弹出式菜单中的“Add Items (By Name)…”，弹出如图 B-52 所示的对话框，在相应的 SFR 中的寄存器名称上双击即可。



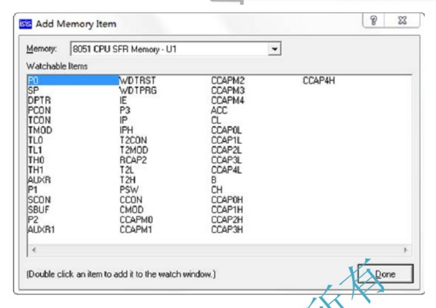


图 B-52　双击鼠标左键添加观察项名称

(2) 以添加地址方式添加观察项。这种方式既可以添加内部 RAM 中的观察项，也可以添加 SFR 中的观察项，如图 B-53 所示。添加 IDATA(内部 RAM)中地址为 0x50 单元、名称为 Stack 的观察项，数据类型为字节 Byte，数据格式为十六进制，再单击窗口右下角的“Add”按钮添加。

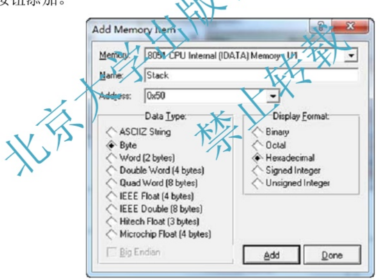


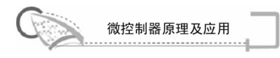
图 B-53　以添加地址方式添加观察项

以地址形式添加 IDATA 观察项时，其地址范围为 0x00~0xFF(十六进制)；以地址形式添加 SFR 观察项时，其地址范围为 80H~FFH。并且要注意所采用的数制表达形式要一致。若用十六进制，形如“0xFF”；若用二进制，形如“0b11111111”，它们都有自己的前缀。否则会弹出警告窗口。若用十进制形式直接输入，则没有前缀和后缀。

2) 删除观察项

在观察窗口单击相应的观察项，按键盘上的“Del”键即可删除该项；或者在观察窗口对某观察项单击鼠标右键，在弹出式菜单中单击“Delete Item”即可。





3) 观察点条件设置

通过观察窗口不仅可以实时查看观察项的值，还可由此设置“观察点条件”触发断点，以满足某些特殊条件断点的要求。在观察窗口内单击鼠标右键，在弹出的快捷菜单中单击“Watchpoint Condition”，弹出观察点条件设置窗口，如图 B-54 所示。

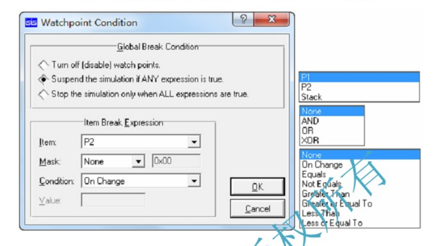


图 B-54　观察点条件设置窗口

观察点条件设置分为两级：

(1) Global Break Condition(全局断点条件设置)，如图 B-54 的上半部分所示。“Turn off (disable) watch points.”(关闭观察点)，“Suspend the simulation if ANY expression is true.”(任何表达式为真时暂停仿真)，“Stop the simulation only when ALL expressions are true.” (所有条件为真，停止仿真)，这 3 个断点条件可选择其中之一。一般选择“Suspend the simulation if ANY expression is true.”。

(2) Item Break Expression(观察项的断点表达式)，如图 B-54 的下半部分所示。其中 Item 为观察窗口中添加的观察项，可单击组合框按钮，在其下拉列表中选择要设置断点的观察项。各个 Item(观察项)及其 Mask(约束条件)、Condition(条件)，如图 B-54 右边所示。当前“P2”观察项的断点条件为“On Change” (改变时)，即当 P2 的值改变时，暂停仿真。

在本实例中，单击图 B-42 中电路的按钮，P2 中的值发生变化，触发断点使仿真暂停，如图 B-55 所示。

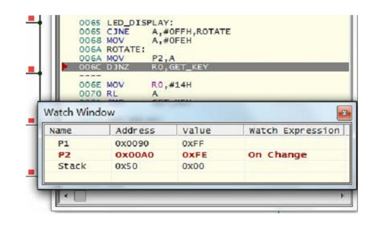


图 B-55　带观察点条件断点触发的仿真





B.3　Proteus 与 Keil μVision4 的联合仿真

Proteus 与 Keil μVision4 的联合仿真非常简单，只需安装一个软件，再将 Proteus 与 Keil μVision4 设置一下即可。

(1) 安装 Proteus 与 Keil μVision4。

(2) 安装 vdmagdi.exe。

(3) 在 Proteus 中建立一个实例。

(4) 在 Keil μVision4 中建立一个项目，并编写程序，将此项目和 Proteus 建立的项目存放在同一个文件夹下。

(5) 在 Keil μVision4 中，“Options for Target ‘Target 1’”对话框的 Debug 选项中，单击选中“Use:”，并在其右边下拉框中选中“Proteus VSM Simulator”；然后单击“Settings”按钮，出现如图 B-56 所示窗口。如果 Proteus 与 Keil μVision4 安装在同一台计算机中，选择默认设置即可。

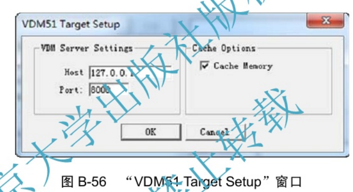


图 B-56　“VDM51 Target Setup”窗口

(6) 在 Proteus 中，单击菜单“Debug”，选中“Use Remote Debug Monitor”项，即在“Use Remote Debug Monitor”前出现“√”即可。

(7) 经过以上设置后，即可在 Keil μVision4 中调试 Proteus 中的 MCU 系统。

附录 C

ASCII 码表(常用)

字符	十六进制	字符	十六进制	字符	十六进制	字符	十六进制
NUL(空)	0	6	36	N	4E	g	67
换行	A	7	37	O	4F	h	68
空格	20	8	38	P	50	i	69
!(感叹号)	21	9	39	Q	51	j	6A
”	22	：	3A	R	52	k	6B
#	23	；	3B	S	53	l	6C
$	24	<	3C	T	54	m	6D
%	25	=	3D	U	55	n	6E
&	26	>	3E	V	56	o	6F
'(引号)	27	?	3F	W	57	p	70
(	28	@	40	X	58	q	71
)	29	A	41	Y	59	r	72
*	2A	B	42	Z	5A	s	73
+	2B	C	43	[	5B	t	74
,	2C	D	44	\	5C	u	75
-(减号)	2D	E	45	]	5D	v	76
.	2E	F	46	^	5E	w	77
/(除号)	2F	G	47	-	5F	x	78
0	30	H	48	a	61	y	79
1	31	I	49	b	62	z	7A
2	32	J	4A	c	63	{	7B
3	33	K	4B	d	64	}	7D
4	34	L	4C	e	65		
5	35	M	4D	f	66		

附录 D

80C51 系列微控制器指令系统表

表 D-1　数据传递类指令表

助记符		指令说明	字节数	周期数
MOV	A,Rn	寄存器传送到累加器	1	1
MOV	A,direct	直接地址传送到累加器	2	1
MOV	A,@Ri	内部间接 RAM 传送到累加器	1	1
MOV	A,#data	立即数传送到累加器	2	1
MOV	Rn,A	累加器传送到寄存器	1	1
MOV	Rn,direct	直接地址传送到寄存器	2	2
MOV	Rn,#data	立即数传送到寄存器	2	1
MOV	direct,Rn	寄存器传送到直接地址	2	1
MOV	direct,direct	直接地址传送到直接地址	3	2
MOV	direct,A	累加器传送到直接地址	2	1
MOV	direct,@Ri	内部间接 RAM 传送到直接地址	2	2
MOV	direct,#data	立即数传送到直接地址	3	2
MOV	@Ri,A	累加器传送到内部间接 RAM	1	2
MOV	@Ri,direct	直接地址传送到内部间接 RAM	2	1
MOV	@Ri,#data	立即数传送到内部间接 RAM	2	2
MOV	DPTR,#data16	16 位常数加载到数据指针	3	1
MOVC	A,@A+DPTR	代码字节传送到累加器	1	2
MOVC	A,@A+PC	代码字节传送到累加器	1	2
MOVX	A,@Ri	外部 RAM(8 位地址)传送到累加器	1	2
MOVX	A,@DPTR	外部 RAM(16 位地址)传送到累加器	1	2
MOVX	@Ri,A	累加器传送到外部 RAM(8 位地址)	1	2
MOVX	@DPTR,A	累加器传送到外部 RAM(16 位地址)	1	2
PUSH	direct	直接地址压入堆栈	2	2
POP	direct	直接地址弹出堆栈	2	2
XCH	A,Rn	寄存器和累加器交换	1	1
XCH	A, direct	直接地址和累加器交换	2	1
XCH	A, @Ri	内部间接 RAM 和累加器交换	1	1
XCHD	A, @Ri	内部间接 RAM 和累加器交换低 4 位字节	1	1

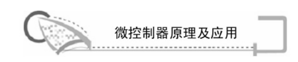


表 D-2 算术运算类指令表

助记符		指令说明	字节数	周期数
INC	A	累加器加 1	1	1
INC	Rn	寄存器加 1	1	1
INC	direct	直接地址加 1	2	1
INC	@Ri	间接 RAM 加 1	1	1
INC	DPTR	数据指针加 1	1	2
DEC	A	累加器减 1	1	1
DEC	Rn	寄存器减 1	1	1
DEC	direct	直接地址减 1	2	2
DEC	@Ri	间接 RAM 减 1	1	1
MUL	AB	累加器和 B 寄存器相乘	1	4
DIV	AB	累加器除以 B 寄存器	1	4
DA	A	累加器十进制调整	1	1
ADD	A,Rn	寄存器与累加器求和	1	1
ADD	A,direct	直接地址与累加器求和	2	1
ADD	A,@Ri	间接 RAM 与累加器求和	1	1
ADD	A,#data	立即数与累加器求和	2	1
ADDC	A,Rn	寄存器与累加器求和(带进位)	1	1
ADDC	A,direct	直接地址与累加器求和(带进位)	2	1
ADDC	A,@Ri	内部间接 RAM 与累加器求和(带进位)	1	1
ADDC	A,#data	立即数与累加器求和(带进位)	2	1
SUBB	A,Rn	累加器减去寄存器(带借位)	1	1
SUBB	A,direct	累加器减去直接地址(带借位)	2	1
SUBB	A,@Ri	累加器减去内部间接 RAM(带借位)	1	1
SUBB	A,#data	累加器减去立即数(带借位)	2	1

表 D-3 逻辑运算类指令表

助记符		指令说明	字节数	周期数
ANL	A,Rn	寄存器“与”到累加器	1	1
ANL	A,direct	直接地址“与”到累加器	2	1
ANL	A,@Ri	内部间接 RAM“与”到累加器	1	1
ANL	A,#data	立即数“与”到累加器	2	1
ANL	direct,A	累加器“与”到直接地址	2	1
ANL	direct, #data	立即数“与”到直接地址	3	2
ORL	A,Rn	寄存器“或”到累加器	1	2
ORL	A,direct	直接地址“或”到累加器	2	1
ORL	A,@Ri	内部间接 RAM“或”到累加器	1	1
ORL	A,#data	立即数“或”到累加器	2	1
ORL	direct,A	累加器“或”到直接地址	2	1

续表

助记符		指令说明	字节数	周期数
ORL	direct, #data	立即数“或”到直接地址	3	1
XRL	A,Rn	寄存器“异或”到累加器	1	2
XRL	A,direct	直接地址“异或”到累加器	2	1
XRL	A,@Ri	内部间接 RAM“异或”到累加器	1	1
XRL	A,#data	立即数“异或”到累加器	2	1
XRL	direct,A	累加器“异或”到直接地址	2	1
XRL	direct, #data	立即数“异或”到直接地址	3	1
CLR	A	累加器清零	1	2
CPL	A	累加器求反	1	1
RL	A	累加器循环左移	1	1
RLC	A	带进位累加器循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	带进位累加器循环右移	1	1
SWAP	A	累加器高、低 4 位交换	1	1

表 D-4　控制转移类指令表

助记符		指令说明	字节数	周期数
JMP	@A+DPTR	相对 DPTR 的无条件间接转移	1	2
JZ	rel	累加器为 0 则转移	2	2
JNZ	rel	累加器为 1 则转移	2	2
CJNE	A,direct,rel	比较直接地址和累加器，不相等转移	3	2
CJNE	A,#data,rel	比较立即数和累加器，不相等转移	3	2
CJNE	Rn,#data,rel	比较寄存器和立即数，不相等转移	2	2
CJNE	@Ri,#data,rel	比较立即数和内部间接 RAM，不相等转移	3	2
DJNZ	Rn,rel	寄存器减 1，不为 0 则转移	3	2
DJNZ	direct,rel	直接地址减 1，不为 0 则转移	3	2
NOP		空操作，用于短暂延时	1	1
ACALL	add11	绝对调用子程序	2	2
LCALL	add16	长调用子程序	3	2
RET		从子程序返回	1	2
RETI		从中断服务子程序返回	1	2
AJMP	add11	无条件绝对转移	2	2
LJMP	add16	无条件长转移	3	2
SJMP	rel	无条件相对转移	2	2

表 D-5　布尔指令表

助记符		指令说明	字节数	周期数
CLR	C	清进位位	1	1
CLR	bit	清直接寻址位	2	1
SETB	C	置位进位位	1	1
SETB	bit	置位直接寻址位	2	1
CPL	C	取反进位位	1	1
CPL	bit	取反直接寻址位	2	1
ANL	C,bit	直接寻址位“与”到进位位	2	2
ANL	C，/bit	直接寻址位的反码“与”到进位位	2	2
ORL	C,bit	直接寻址位“或”到进位位	2	2
ORL	C，/bit	直接寻址位的反码“或”到进位位	2	2
MOV	C,bit	直接寻址位传送到进位位	2	1
MOV	bit, C	进位位传送到直接寻址位	2	2
JC	rel	如果进位位为 1 则转移	2	2
JNC	rel	如果进位位为 0 则转移	2	2
JB	bit,rel	如果直接寻址位为 1 则转移	3	2
JNB	bit,rel	如果直接寻址位为 0 则转移	3	2
JBC	bit,rel	直接寻址位为 1 则转移并清除该位	2	2

表 D-6　伪指令表

助记符	指令说明
ORG	指明程序的开始位置
DB	定义数据表
DW	定义 16 位的数据表
EQU	给一个表达式或一个字符串起名
DATA	给一个 8 位的内部 RAM 起名
XDATA	给一个 8 位的外部 RAM 起名
BIT	给一个可位寻址的位单元起名
END	指出源程序到此为止

表 D-7　指令中的符号标识表

助记符	指令说明
Rn	工作寄存器 R0～R7
Ri	工作寄存器 R0 和 R1
@Ri	间接寻址的 8 位 RAM 单元地址(00H～FFH)
#data8	8 位常数

续表

助记符	指令说明
#data16	16 位常数
addr16	16 位目标地址，能转移或调用到 64KBROM 的任何地方
addr11	11 位目标地址，在下条指令的 2KB 范围内转移或调用
Rel	8 位偏移量，用于 SJMP 和所有条件转移指令，范围-128～+127
Bit	片内 RAM 中的可寻址位和 SFR 的可寻址位
Direct	直接地址，范围片内 RAM 单元(00H～7FH)和 SFR 区的 80H～FFH
$	指本条指令的起始位置

